P.1877/80

4

1990

informatyka

Okres istnienia oprogramowania

Język Lotos

WYDAWNICTWO NOT SIGMA

Cena 5800 zł

informatyka

Miesięcznik

ISSN 0542-9951. INDEKS 36124

nr 4 1990
Kwiecień
Rok wydania XXV

W numerze:

Strona

System MikroAPO. Podsystem projektowania technologii dla typoszeregów części
Jacek Kardasiewicz 1

Architektury procesorów tablicowych tolerujących błędy
Ilona Bluemke 5

Środowisko programowe do projektowania pakietów elektronicznych systemów mikroprocesorowych
Sławomir Czepielewski, Zdzisław Nowakowski 11

Okres istnienia oprogramowania (3)
Janusz Zalewski 14

Propozycja nowego programu uniwersyteckich studiów informatycznych (3)
Lech Banachowski, Jarosław Deminet, Marek J. Lao 21

Ze świata 24
Symboliczny program uruchomieniowy dla Ady (1)
Nowa generacja sieci lokalnych
Tranzystory kwantowe
Trzy systemy programowania w Moduli 2
Oprogramowanie systemu sieciowego IBM

Terminologia III okł.
Atrybuty jakości oprogramowania – definicje podręcznikowe

W najbliższych numerach:

- Lesław Sieniawski opisuje genezę komputerów o architekturze zredukowanej – RISC, przedstawia wnioski z eksperymentów badawczych nad ulepszeniem architektury maszyn oraz kreśli ich historię.
- Maria Bronowska omawia system graficzny Ramtek RM-9460 – wieloprocesorowe urządzenie graficzne, które dekoduje i przetwarza zbiór rozkazów wysokiego poziomu wysyłany z komputera centralnego.
- Henryk J. Runka przedstawia w swoim artykule „Duża Macierz w Turbo Pascalu 5.0" w jaki sposób operować macierzą zajmującą większą pamięć niż 64Kb.
- Jacek Malec charakteryzuje wyspecjalizowane stacje robocze przeznaczone do przetwarzania symbolicznego w języku Lisp.
- Tomasz Żakowicz prezentuje pakiet CDS/ISIS umożliwiający tworzenie i obsługę systemów baz danych.
- Piotr Dembiński omawia normy w projektowaniu sieci komputerowych i protokołów komunikacyjnych.

KOLEGIUM REDAKCYJNE

Mgr Jarosław DEMINET,
dr inż. Wacław ISZKOWSKI,
mgr Teresa JABŁOŃSKA
(sekretarz redakcji),
Władysław KLEPACZ
(redaktor naczelny),
dr inż. Marek MACHURA,
dr inż. Wiktor RZECZKOWSKI,
mgr inż. Jan RYŻKO,
mgr Hanna WŁODARSKA,
dr inż. Janusz ZALEWSKI,
(zastępca redaktora naczelnego).

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

Wydawca: Wydawnictwo Czasopism i Książek Technicznych SIGMA NOT
Spółka z o.o.
ul. Biała 4,
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja: 01-552 Warszawa,
Pl. Inwalidów 10, p. 128, 136
tel. 39-14-34

Materiałów nie zamówionych redakcja nie zwraca

Druk:
PRASOWE ZAKŁADY GRAFICZNE
RSW „PRASA-KSIĄŻKA-RUCH"
ul. Dworcowa 13,
85-950 BYDGOSZCZ
Zam. 564/90
Obj. 4,0 ark. druk. Nakład 6000 egz.

Cena egzemplarza 5800 zł

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji

WARUNKI PRENUMERATY

Prenumeratorzy zbiorowi – jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia). Blankiety te będą dostarczane dotychczasowym prenumeratorom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu, w Radach Wojewódzkich NOT bądź w Redakcjach czasopism.

Prenumeratorzy indywidualni – osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: **Państwowy Bank Kredytowy III/O Warszawa nr 370015-7490-139-11.**

Prenumerata ulgowa – przysługuje wyłącznie osobom fizycznym – członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

Prenumerata ze zleceniem wysyłki za granicę – zamawia się tak, jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.

Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Wpłaty na prenumeratę przyjmowane są w terminach:
- do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
- do 28 lutego na II, III i IV kwartał oraz II półrocze,
- do 31 maja na III i IV kwartał oraz II półrocze,
- do 31 sierpnia na IV kwartał.

Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela Zakład Kolportażu Wydawnictwa NOT SIGMA (ul. Bartycka 20, 00-716 Warszawa), skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism – można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa ul. Mazowiecka 12 (tel. 26-80-16), lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena egzemplarza: normalna 5800 zł, ulgowa 1160 zł

Uwaga! Podane ceny mają charakter wstępny i mogą ulec zmianie, w związku z powyższym Wydawnictwo zastrzega sobie prawo żądania dopłat do już opłaconej prenumeraty.

Poniższy artykuł został zamówiony po pokazie programu na wystawie „Infosystem'89", jako przykład praktycznego rozwiązania znajdującego sporą liczbę nabywców. Szczególnie dobre wrażenie zrobił wtedy edytor graficzny, szybki i prosty w obsłudze – wiadomo przecież, jak trudno stworzyć od zera interakcyjny program manipulowania grafiką. Poza tym, przeczy on często spotykanej zasadzie, według której programy inżynierskie, tworzone „dla mnie samego i dla moich kolegów", wymagają wielkiego doświadczenia przy obsługiwaniu. Opisany program wydaje się bardzo przyjazny. Warto przy okazji dodać, że system APO jest chyba zbyt mało znany, a to przecież jeden z niewielu programów już od wielu lat rozprowadzanych handlowo w Polsce. Może uda nam się jeszcze wrócić do tej tematyki.
(J. D.)

POLITECHNIKA ŚLĄSKA Biblioteka Główna
R 1877/90

JACEK KARDASIEWICZ
Soft Produkt Sp. z o.o.
Warszawa

System MikroAPO

Podsystem projektowania technologii dla typoszeregów części

Problematyka komputerowego wspomagania projektowania (KWP) gości na łamach *Informatyki* dość rzadko, pomimo iż potencjalnie stanowi obszar działania dla daleko większej grupy informatyków niż inne dziedziny. Działalność ta stawia jednak przed informatykami wiele dodatkowych wymagań, związanych z gruntownym poznaniem innych dziedzin wiedzy (np. technologii maszyn), a także sprowadza informatykę do roli narzędzia umożliwiającego tworzenie oprogramowania, które musi być bezwzględnie podporządkowane celowi nadrzędnemu, jakim jest optymalne projektowanie. Praktyka wykazuje, że wykonanie takiego oprogramowania jest niemożliwe bez opanowania przez informatyka reguł projektowania bez użycia komputera.

Jedną z wielu dziedzin, w których zastosowanie systemów komputerowego wspomagania projektowania przynosi wymierne efekty jest projektowanie procesów technologicznych części maszyn.

SYSTEM MikroAPO i ZREALIZOWANY PODSYSTEM

System MikroAPO stanowi mikrokomputerową wersję systemu APO, którego historia liczy już ponad 17 lat. Przez wersje opracowywane kolejno dla komputerów Odra 1204, Odra 1300, Riad 32, PDP 11/34 i SM 4 zespoły autorskie opracowały wersję systemu przeznaczoną dla mikrokomputerów zgodnych z IBM PC. Wraz ze sprzętem komputerowym zmieniał się także zakres funkcji systemu, jego organizacja oraz rozwiązania informatyczne. Obecnie obszar zastosowań systemu MikroAPO obejmuje oprócz programowania obrabiarek sterowanych numerycznie także symulacją ich pracy, projektowanie kompletnych procesów technologicznych i emisję dokumentacji warsztatowej, optymalizację rozkroju blach, projektowanie krzywek sterujących pracą automatów tokarskich oraz konstrukcję i projektowanie technologii przekładni zębatych. Lista ta jest uzupełniana w miarę zapotrzebowania zakładów przemysłowych.

System MikroAPO jest podzielony na podsystemy, stanowiące jednostki zwarte pod względem funkcjonalnym (Podsystem programowania tokarek sterowanych numerycznie, Podsystem optymalizacji rozkroju gilotynowego blach, czy Podsystem projektowania technologii dla typoszeregów części). Dzięki modułowi zarządzającemu pracą systemu, wszystkie te podsystemy tworzą jedną całość i mogą wzajemnie wykorzystywać swoje dane i wyniki pracy.

System MikroAPO jest oryginalnym polskim rozwiązaniem rozwijanym przez następujące instytucje: Ośrodek Badawczo-Konstrukcyjny KOPROTECH, ZETO Jelenia Góra, Instytut Technologii Mechanicznej, Instytut Matematyki i Instytut Podstaw Budowy Maszyn Politechniki Warszawskiej, Spółki CEKOMP i SoftProdukt, a także prywatną pracownię informatyczną autora artykułu. Prace finansowane są częściowo ze środków Centralnego Programu Badawczo-Rozwojowego 7.5, koordynowanego przez OBK KOPROTECH, a częściowo ze środków własnych wymienionych zespołów autorskich. Historia systemu APO notuje kilkadziesiąt wdrożeń przemysłowych, m.in. w takich zakładach, jak: CHEMAR Kielce, PZO Warszawa, ZAMECH Elbląg, WSK PZL Świdnik i Huta Warszawa.

Poniżej zaprezentowano opracowany w pracowni autora, Podsystem projektowania technologii dla typoszeregów części, stanowiący jeden z elementów składowych systemu MikroAPO.

Wymagania użytkowe dla systemów KWP

Eksploatowane w Polsce systemy KWP zawierają niewielki zestaw środków pozwalających na przystosowanie ich do konkretnych warunków przemysłowych. Cecha ta stanowi ich poważną wadę, szczególnie ze zwględu na istnienie dużej różnorodności zagadnień technologicznych i metod ich rozwiązywania w poszczególnych przedsiębiorstwach przemysłowych.

Indywidualizacja procesów technologicznych wynika z wielu czynników. Są to przede wszystkim:
- poziom techniczny przedsiębiorstwa,
- baza obrabiarkowa przedsiębiorstwa,
- baza narzędziowa przedsiębiorstwa,
- konieczność przestrzegania norm zakładowych i branżowych,
- przyzwyczajenia kadry technologicznej.

Wszystkie te czynniki charakteryzuje duża zmienność w zależności od rozpatrywanych przedsiębiorstw i czasu.

Z punktu widzenia technologa, w praktyce przydatny jest tylko taki system KWP, który pozwala na opracowywanie optymalnych rozwiązań w warunkach konkretnego zakładu przemysłowego. Optymalność ta nie jest jednak ściśle zdefiniowana i zależy często od subiektywnych poglądów i przyzwyczajeń technologa oceniającego system.

Istnieją dwie klasy wymagań stawianych przez użytkowników systemom KWP. Są to wymagania obiektywne i wymagania subiektywne. Spełnienie wymagań obiektywnych warunkuje wykorzystanie systemu KWP. Wymagania subiektywne powinny być spełnione ze względu na efektywność wspomagania projektowania i wygodę użytkowników, choć nie jest to warunkiem koniecznym. Wymagania obiektywne to między innymi:
- możliwość realizacji za pomocą systemu KWP niezbędnych – z punktu widzenia konkretnego użytkownika – zadań projektowych,
- poprawność odwzorowania praw fizyki przez algorytmy projektowe systemu,
- zgodność algorytmów projektowych z przepisami obowiązującymi projektanta,
- dopasowanie otrzymywanych wyników procesu projektowania do warunków konkretnego przedsiębiorstwa.

Do wymagań subiektywnych można zaliczyć:
- możliwość efektywnego wykorzystywania tej samej wersji systemu KWP przez użytkowników o różnym poziomie wiedzy,
- możliwość wyboru wariantu procesu technologicznego i nakładanie na projektowanie ograniczeń, wynikających z subiektywnych kryteriów stosowanych w warunkach konkretnego przedsiębiorstwa,
- stosowanie terminologii i form danych, uznanych w konkretnym środowisku za obowiązujące,
- stosowanie przyjętych w przedsiębiorstwie wzorów dokumentów,

● możliwość wyboru zasad wymiany informacji pomiędzy użytkownikiem a systemem KWP (dialog, dane wsadowe, graficzne wprowadzanie informacji itp.).

Prawidłowo zaprojektowany sytem KWP musi uwzględnić możliwość adaptowania go do konkretnych warunków eksploatacji, w tym także do subiektywnych nawyków eksploatujących go technologów.

Przedstawione powyżej wymagania stawiane przez użytkowników systemom KWP jednoznacznie wskazują, że systemy te winny być systemami wspomagającymi projektowanie, a nie projektującymi fragmenty procesów technologicznych w sposób automatyczny, co potwierdzają liczne – zakończone niepowodzeniem – próby tworzenia i wdrażania systemów projektujących w sposób automatyczny. Powyższy wniosek stanowi podstawę projektu Podsystemu projektowania technologii dla typoszeregów części.

Funkcje podsystemu

Podsystem projektowania technologii dla typoszeregów części wykorzystuje metodę projektowania parametrycznego na bazie typowego procesu technologicznego i typowej części. Wynika stąd, że ostateczna postać projektu procesu technologicznego zadanej części jest uzyskiwana na drodze konkretyzacji ogólnionego (parametrycznego) projektu procesu technologicznego. Konkretyzacja ta polega na obliczaniu wartości wyrażeń arytmetycznych i konwersji wyników na postać znakową oraz na interpretacji zdań prostego języka programowania.

Przyjęto, że każdorazowo przed rozpoczęciem projektowania technologii dla nowego typoszeregu części, użytkownik musi zdefiniować klasę części należących do typoszeregu oraz uogólniony proces technologiczny dla tej klasy i dopiero po jego zdefiniowaniu może przystąpić do generowania procesów technologicznych dla konkretnych części należących do tego typoszeregu. Praktyka wskazuje, że definiowanie uogólnionych procesów technologicznych za pomocą prezentowanego podsystemu nie sprawia technologom trudności formalnych. Problem stanowią jedynie zagadnienia natury technologicznej, które zgodnie z faktami przedstawionymi powyżej, mogą być rozwiązane w sposób optymalny tylko przez technologów w konkretnym zakładzie przemysłowym. System KWP udostępnia jedynie efektywne narzędzia wspomagające realizację tego zadania.

W wyniku przyjętej metody projektowania funkcje podsystemu zostały podzielone na dwie grupy:
1. Definiowanie uogólnionego procesu technologicznego:
● edycja plików tekstowych (edytor tekstów),
● edycja rysunków (edytor graficzny),
● interakcyjne definiowanie scenariuszy dialogów,
● kompilacja definicji scenariuszy dialogów,
● wyświetlanie zawartości plików na ekranie,
● usuwanie plików,
2. Projektowanie procesów technologicznych części (konkretyzacja uogólnionego procesu technologicznego).

Druga grupa funkcji, w przeciwieństwie do grupy pierwszej, generuje projekty procesów technologicznych w sposób automatyczny. Istotne znaczenie ma jednak fakt, że reguły tego automatycznego projektowania są całkowicie zgodne z wymaganiami użytkownika, a nie narzucone przez autora podsystemu.

Definiowanie uogólnionego procesu technologicznego

W celu zdefiniowania uogólnionego procesu technologicznego dla uogólnionej części należy kolejno:
– określić klasę części tworzących typoszereg,
– zastąpić wybrane stałe (wymiary przedmiotu, ich tolerancje, parametry obróbki itp.) zmiennymi,
– zdefiniować scenariusz dialogu służącego do wprowadzania faktycznych wartości zmiennych,
– zdefiniować wzorcowy rysunek przedmiotu, kreślony na ekranie monitora podczas dialogu,
– zapisać specyfikację zadania projektowego,
– zapisać kolejno wszystkie dane należące do definicji uogólnionego procesu technologicznego (opisy rysunków i dokumentów, dane dla innych programów).

Podczas definiowania uogólnionych rysunków, dokumentów i danych dla innych programów użytkownik może używać prostego algorytmicznego języka programowania i wyrażeń arytmetycznych zapisywanych zgodnie z powszechnie przyjętą notacją (nawiasy, znaki, działań arytmetycznych i logicznych, funkcje standardowe itp.). Wszystkie pliki są plikami tekstowymi, a sekwencje zdań języka programowania i wyrażenia arytmetyczne występują w tych plikach jako „wtrącenia" ujmowane w nawiasy klamrowe i kwadratowe.

Ponieważ wszystkie pliki są plikami tekstowymi, więc jest możliwe ich redagowanie za pomocą dowolnego edytora tekstów. Ponadto, dla wygody użytkownika, podsystem oferuje narzędzia służące do interakcyjnego definiowania scenariuszy dialogów i rysunków. W obu przypadkach wynikiem są pliki tekstowe zawierające dane zapisane w językach symbolicznych.

SPECYFIKACJA ZADANIA PROJEKTOWEGO

Zadanie projektowe jest specyfikowane oddzielnie dla każdego typoszeregu części za pomocą edytora tekstów (podsystem dopuszcza zdefiniowanie 90 różnych typoszeregów). Wszelkie nazwy plików są lokalne dla każdej specyfikacji zadania. Poniżej podano przykłady specyfikacji zadań:

● Polecenie konkretyzacji informacji zawartych w pliku o nazwie *filename1* i zapisanie wyników w pliku o nazwie *filename2*

Przekształć *filename1* w *filename2*

● Polecenie narysowania na zadanym urządzeniu rysunku zapisanego w pliku o nazwie *filename* (kreślenie na ploterze jest realizowane za pośrednictwem pakietu AutoCAD)

Narysuj *filename* na ekranie
Narysuj *filename* na drukarce
Narysuj *filename* na ploterze

● Polecenie wyświetlenia na ekranie monitora danych tekstowych zapisanych w pliku o nazwie *filename*

Wyświetl *filename*

● Polecenie wydrukowania na drukarce danych tekstowych zapisanych w pliku o nazwie *filename*

Drukuj *filename*

● Polecenie usunięcia pliku o nazwie *filename*

Usuń *filename*

● Polecenie wykonania programu lub pliku poleceń systemu MS-DOS o nazwie *filename*

Wykonaj *filename*

● Polecenie zakończenia zadania projektowego

Koniec.

Za pomocą wymienionych zdań użytkownik precyzuje sekwencję czynności, które muszą być wykonane w celu zaprojektowania procesu technologicznego części.

Przykładowa specyfikacja zadania ma postać następującą:

```
Przekształć WYKROJ.PIC w WYKROJ
Narysuj WYKROJ na ploterze
Przekształć WYKROJ.DAT w \PROGRAMY\PRASA\DANE.DXF
Wykonaj PRASA.BAT
Przekształć WYKROJ.TEX w WYKROJ
Wyświetl WYKROJ
Wydrukuj WYKROJ
Usuń WYKROJ
Koniec
```

W powyższym przykładzie kolejne polecenia powodują:
● skonkretyzowanie uogólnionego (parametrycznego) rysunku konstrukcyjnego (plik WYKROJ.PIC) i wygenerowanie konkretnego rysunku konstrukcyjnego (plik WYKROJ),
● wykreślenie rysunku konstrukcyjnego (plik WYKROJ),

- skonkretyzowanie uogólnionego zbioru danych (plik WYKROJ.DAT) do postaci zbioru danych (plik DANE.DXF) dla programu PRASA,
- wykonanie pliku poleceń systemu MS-DOS (plik PRASA.BAT),
- skonkretyzowanie uogólnionego opisu procesu technologicznego (plik WYKROJ.TEX) do postaci konkretnego procesu (plik WYKROJ),
- wyświetlenie na ekranie procesu technologicznego (plik WYKROJ),
- wydrukowanie procesu technologicznego (plik WYKROJ),
- usunięcie zbędnego pliku danych (plik WYKROJ).

Wykorzystywany w przykładowym zadaniu plik poleceń PRASA.BAT ma postać:

```
ECHO OFF
CD \PROGRAMY\PRASA
PRASA DANE.DXF
CD \MIKROAPO\TECH
ECHO ON
```

WYKONANIE ZADANIA PROJEKTOWEGO

Wykonanie zadania projektowego polega na wybraniu typoszeregu i przeprowadzeniu dialogu według zdefiniowanego wcześniej scenariusza, w celu wprowadzenia aktualnych wartości parametrów charakteryzujących konkretną część. Po zakończeniu dialogu rozpoczyna się realizacja zadania projektowego, zgodnie z jego specyfikacją. Kolejno konkretyzowane są dane zapisane w plikach, wykonywane rysunki, wyświetlane i drukowane dokumenty. W efekcie użytkownik uzyskuje proces technologiczny części, zgodny ze specyfikacją zadania projektowego.

Definiowanie scenariusza dialogu

Definiowanie scenariusza dialogu zazwyczaj poprzedza definiowanie uogólnionego procesu technologicznego. Użytkownik ma do dyspozycji dwa narzędzia, które pozwalają mu definiować scenariusze dialogu. Pierwszym z nich jest interakcyjny moduł definiowania scenariuszy, który generuje pliki danych zawierające definicje zapisane w specjalnie zaprojektowanym języku, dzięki analizie odpowiedzi użytkownika na zadawane pytania i położenia punktów wskazywanych przez niego za pomocą myszki. Interakcyjne definiowanie scenariuszy dialogów jest najprostszą metodą postępowania. Ponadto użytkownik ma do dyspozycji edytor tekstów oraz pełny opis języka. Definiowanie scenariuszy dialogów za pomocą tego języka jest zdecydowanie bardziej kłopotliwe dla niedoświadczonego użytkownika, ale w przypadku bardziej złożonych typoszeregów pozwala na wykorzystanie wszystkich środków oferowanych przez podsystem. Możliwe jest oczywiście przygotowanie wstępnej wersji definicji w sposób interakcyjny, a następnie uzupełnienie jej za pomocą edytora tekstów i pełnej wersji języka.

Przyjęto, że bezpośrednio po zakończeniu definiowania scenariusza dialogu, źródłowa postać definicji przekształcana jest automatycznie na postać binarną, umożliwiającą znaczne zwiększenie szybkości dialogu podczas wykonywania zadań projektowych. Działanie to zostało nazwane kompilacją definicji scenariusza dialogu.

```
cursor blob off
type \(c(1,1))(x+) ____________Podsystem__projektowania   &
__technologii__dla__typoszeregów__części_____________ (x-)\
frame 11  r51 t27 b30 c2
frame 11  r89 t2  b25 c2
frame 152 r89 t26 b34 c2
frame 11  r51 t31 b34 c2
def form nazwa \ (c(32,3))Nazwa_przedmiotu_:_(p(33,3)s(47)1\47)\
def form dane  \                                              &
(c(27,54))Średnica_D1________=___(p(27,75)r(20.0,)05\7) &
(c(28,54))Średnica_D2________=___(p(28,75)r(10.0,)12\7) &
(c(29,54))Średnica_D3________=___(p(29,75)r(10.0,)19\7) &
(c(30,54))Długość_wałka_L1___=___(p(30,75)r(30.0,)26\7) &
(c(31,54))Długość_stopnia_L2_=___(p(31,75)r(30.0,)33\7) &
(c(32,54))Długość_stopnia_L3_=___(p(32,75)r(30.0,)40\7) &
(c(33,54))Długość_fazy_F_____=___(p(33,75)r(0.05,)47\7) &
                                 (p(27,83)s(4)54\4)      &
                                 (p(28,83)s(4)61\4)      &
                                 (p(29,83)s(4)68\4)      \
type \(c(26,1))(x+)              &
_Nazwa_typoszeregu_:_VALEK       &
_(_Valek_trzystopniowy_)_(x-)    &
(c(28,2))_Varunki_:_             &
(c(28,18))_D1_)_D2__i___D1_)_D3_&
(c(29,18))_L1_)_L2_+_L3_+_2_*_F_\
type \(c(3,2))_Rysunek_przedmiotu_:_\
function 1
cursor blob on
form nazwa
function 2
set 01 1 10
set 02 s RA=____RB=____RC=____RD=____RE=____    &
         RF=____RG=____SA=____SB=____SC=____
form dane
function 3
function 4
end
```

Wydruk 1

Na wydruku 1 zamieszczono przykładową definicję scenariusza dialogu w wersji źródłowej. Tak zapisana definicja wymaga już tylko przetworzenia na postać binarną i może być wykorzystywana w procesie wymiany informacji pomiędzy użytkownikiem, a podsystemem.

Definiowanie dokumentów

Podsystem umożliwia dokonywanie „wtrąceń" do dowolnych plików tekstowych. „Wtrącenia" te zawierają zdania algorytmicznego języka programowania i reguły konwersji danych. Konkretyzacja danych polega wówczas na interpretowaniu tych „wtrąceń". Metoda ta pozwala na przygotowanie wzorów dokumentów w wymaganej postaci bezpośrednio przez użytkowników. Następnie definicje te są uzupełniane algorytmami, które mają na celu wstawienie do dokumentów konkretnych, obliczonych wartości.

Zapewnione przez podsystem środki, w przypadku projektowania dokumentów występujących w procesach technologicznych części maszyn, są zazwyczaj wystarczające. Jeśli konieczne jest dokonanie bardziej skomplikowanych obliczeń (np. zaprogramowanie obrabiarki sterowanej numerycznie), to możliwe jest skorzystanie ze zdania *WYKONAJ...* w specyfikacji zadania projektowego. Dokumenty mogą być także przygotowywane w postaci graficznej. Są one wówczas traktowane tak jak rysunki i muszą w całości mieścić się na ekranie monitora. W celu zobrazowania przyjętej konwencji, na wydruku 2 przytoczono przykład definicji ramowego procesu technologicznego części.

```
          code A(1)=A$
          code A(5)=B$
          code A(9)=C$
          X=0
          Y=0
          Z=0
          if A(1)<>0 then X=0.2
          if A(5)<>0 then Y=0.2
          if A(9)<>0 then Z=0.2
)
=============================================================================
|                    P R O C E S   T E C H N O L O G I C Z N Y               |
=============================================================================
|        Nazwa przedmiotu : [<L50)%name]|
|        Opracowano dnia  : [<L50)%date]|
-----------------------------------------------------------------------------
| Operacja 01 | Ciąć materiał na wymiar:          [<R10)D+5]                 |
-----------------------------------------------------------------------------
| Operacja 02 | Toczyć średnice na wymiar:                                   |
|             |                                   [<R10)A+X]                 |
|             |                                   [<R10)B+Y]                 |
|             |                                   [<R10)C+Z]                 |
-----------------------------------------------------------------------------
| Operacja 03 | Toczyć czoła na wymiar:           [<R10) D ]                 |
|             | Toczyć fazy na wymiar:            [<R7>G]/45                 |
(         I=4
          if (A(1)=0) * (A(5)=0) * (A(9)=0) then goto et3
          I=5
)
-----------------------------------------------------------------------------
|             | Szlifować średnice na wymiar:                                |
(         if A(1)=0 then goto et1
)
|             |                                   [<R10)A] [<L4>A$]          |
(    et1: if A(5)=0 then goto et2
)
|             |                                   [<R10)B] [<L4>B$]          |
(    et2: if A(9)=0 then goto et3
)
|             |                                   [<R10)C] [<L4>C$]          |
(    et3:
)
-----------------------------------------------------------------------------
| Operacja 0[I] | Ostre krawędzie stępić                                     |
-----------------------------------------------------------------------------
| Operacja 0[I+1] | Cynkować                                                 |
=============================================================================
```

Wydruk 2

Definiowanie rysunków

Rysunki, podobnie jak scenariusze dialogów, mogą być przygotowywane zarówno za pomocą interakcyjnego edytora graficznego, jak też za pomocą edytora tekstów. Zdecydowanie łatwiejsze jest korzystanie z edytora graficznego, który w przeciwieństwie do modułu definiującego w sposób interakcyjny scenariusze dialogów, pozwala na wykorzystywanie pełnej wersji języka służącego do definiowania rysunków. Jedynym wyjątkiem są te uogólnione rysunki, które wymagają użycia algorytmicznego języka programowania.

Edytor graficzny nie jest wzorowany na powszechnie znanym pakiecie AutoCAD. Wynika to z faktu, że musi on umożliwiać generowanie rysunków uogólnionych (a nie konkretnych), tj. rysunków całych klas części. W projekcie edytora przyjęto metodę definiowania linii konstrukcyjnych rysunku, których położenie może być określane za pomo-

cą wyrażeń arytmetycznych. Kreślenie podstawowych elementów graficznych wymaga uprzedniego zdefiniowania linii konstrukcyjnych. Elementy są wówczas oparte na punktach przecięć linii konstrukcyjnych. Metoda ta ułatwia szybkie przewymiarowywanie rysunków podczas ich definiowania, a także analizę wykonalności rysunków dla różnych zestawów parametrów. Na wydruku 3 podano przykład uogólnionej definicji wałka o trzech stopniach.

```
Line (0,0),(0,168),(240,168),(240,0),(0,0)
Line (0,12),(240,12)
Text w34 c6 'Nazwa przedmiotu: [%name]'
Line (200,12),(200,0)
Text w34 c79 '[%date]'
(               print "Generowanie rysunku wałka."
                print "--------------------------"
    rem ----- Obliczenie skali rysunku -----
                X=240/(D+40)
                Y=158/(A+50)
                S=X
                if Y < X then S=Y
)
Base ([120-(20+D/2)*S],89)
Scale [S]
Line type 1/3
Line (10,0),([30+D],0)
(               print "Generowanie zarysu wałka."
)
Line type 3/1
Line (20,[B/2-G]),([20+G],[B/2]),([20+E],[B/2]),([20+E],[A/2-G]),([20+E+G],[A/2]),&
     ([20+D-F-G],[A/2]),([20+D-F],[A/2-G]),([20+D-F],[C/2]),([20+D-G],[C/2]),      &
     ([20+D],[C/2-G]),([20+D],[-C/2+G]),([20+D-G],[-C/2]),([20+D-F],[-C/2]),       &
     ([20+D-F],[-A/2+G]),([20+D-F-G],[-A/2]),([20+E+G],[-A/2]),([20+E],[-A/2+G]), &
     ([20+E],[-B/2]),([20+G],[-B/2]),(20,[-B/2+G]),(20,[B/2-G])
Line ([20+G],[B/2]),([20+G],[-B/2])
Line ([20+E],[B/2]),([20+E],[-B/2])
Line ([20+E+G],[A/2]),([20+E+G],[-A/2])
Line ([20+D-F-G],[A/2]),([20+D-F-G],[-A/2])
Line ([20+D-F],[C/2]),([20+D-F],[-C/2])
Line ([20+D-G],[C/2]),([20+D-G],[-C/2])
(               print "Generowanie linii wymiarowych."
)
Line type 1/1
Line (20,0),(20,[-A/2-22])
Line ([20+E],0),([20+E],[-A/2-12])
Line ([20+D-F],0),([20+D-F],[-A/2-12])
Line ([20+D],0),([20+D],[-A/2-22])
Line dim ([20+E+(D-E-F)/2+5],[-A/2]),([20+E+(D-E-F)/2+5],[A/2]) c3 h5 '#[A][A$]'
Line dim ([20+E/2+5],[-B/2]),([20+E/2+5],[B/2]) c3 h5 '#[B][B$]'
Line dim ([20+D-F/2+5],[-C/2]),([20+D-F/2+5],[C/2]) c3 h5 '#[C][C$]'
Line dim (20,[-A/2-20]),([20+D],[-A/2-20]) c3 h5 '[D]'
Line dim (20,[-A/2-10]),([20+E],[-A/2-10]) c3 h5 '[E]'
Line dim ([20+D-F],[-A/2-10]),([20+D],[-A/2-10]) c3 h5 '[F]'
Line ([20+D-G],[C/2]),([15+D-G],[C/2+5]),([30+D-G],[C/2+5])
String ([30+D-G],[C/2+5]) h5 a0 j1 '[G]/45'
Line ([20+D-F-G],[A/2]),([15+D-F-G],[A/2+5]),([30+D-F-G],[A/2+5])
String ([30+D-F-G],[A/2+5]) h5 a0 j1 '[G]/45'
Line ([20+E+G],[A/2]),([25+E+G],[A/2+5]),([10+E+G],[A/2+5])
String ([10+E+G],[A/2+5]) h5 a0 j1 '[G]/45'
Line ([20+G],[B/2]),([25+G],[B/2+5]),([10+G],[B/2+5])
String ([10+G],[B/2+5]) h5 a0 j1 '[G]/45'
End
(               print "Rysunek wałka wygenerowano."
)
```

Wydruk 3

Podsystem projektowania technologii dla typoszeregów części umożliwia wykorzystywanie w zadaniach projektowych dowolnych programów, które nie są częścią składową podsystemu. Mogą to być dowolne podsystemy systemu MikroAPO, a także systemy generujące taśmy sterujące pracą obrabiarek sterowanych numerycznie (np. GTL, EXAPT) lub specyficzne programy optymalizujące wyniki obliczeń. Jedynym wymaganiem stawianym wykorzystywanym programom jest, aby mogły one odczytywać dane zapisane w plikach tekstowych i w tej samej postaci zapisywać wyniki swej pracy. Podsystem projektowania technologii dla typoszeregów części generuje wówczas dane dla tych programów i emituje wyniki ich pracy.

W przypadku bardziej złożonych procesów obliczeniowych możliwe jest wykorzystywanie poleceń systemu MS-DOS zapisanych w plikach typu BAT co pozwala na rozmieszczanie programów i plików danych na różnych dyskach logicznych i w różnych katalogach.

Podsumowując można stwierdzić, że zaprezentowany podsystem systemu MikroAPO jest pakietem ukierunkowanym na edycję i interpretację zadań możliwych do parametryzacji. Nie jest on pakietem projektującym procesy technologiczne w sposób automatyczny, lecz stanowi narzędzie ułatwiające technologom pracę. Podsystem jest związany z projektowaniem technologii maszyn jedynie przez swą zewnętrzną postać i stosowane słownictwo. W chwili obecnej trwają prace nad dostosowaniem podsystemu do wspomagania projektowania w dziedzinach innych też technologia maszyn. Jednym z zastosowań będzie, na przykład, projektowanie technologii chemicznych.

W niedalekiej przyszłości Podsystem projektowania technologii dla typoszeregów części będzie umożliwiał między innymi:

- dowolne definiowanie przez użytkowników alfabetów i graficznych wzorów znaków na ekranie monitora, drukarce i w pakiecie AutoCAD,
- współpracę ze wszystkimi dostępnymi w kraju sterownikami graficznymi,
- posługiwanie się kolorem, co powinno znacznie ułatwić przygotowywanie rysunków,
- odczytywanie informacji zapisanych w plikach baz danych dBase i Informix podczas edycji plików tekstowych i rysunków.

Ponadto kierunki rozwoju podsystemu będą wyznaczane przez potrzeby użytkowników i postulaty formułowane przez nich podczas wdrożeń przemysłowych.

Konkurs PTI

W dniu 19 grudnia 1989 roku nastąpiło rozstrzygnięcie wyników kolejnego **VI Ogólnopolskiego Konkursu Polskiego Towarzystwa Informatycznego** na najlepsze prace magisterskie z dziedziny informatyki. Jury Konkursu w składzie: doc. dr inż. Czesław Daniłowicz (przewodniczący), doc. dr hab. Piotr Dembiński, doc. dr hab. Maciej Sysło, dr inż. Zbigniew Szpunar (sekretarz) i prof. dr hab. Władysław M. Turski – po uwzględnieniu opinii recenzentów prac konkursowych i po dyskusji – powzięło następujące decyzje:

Pierwszej nagrody nie przyznano.

Dwie równorzędne **drugie nagrody** po 130 tys. zł otrzymują:

Przemysław Grzeszczak i **Maciej Idzikowski** za pracę pt. *„Realizacja prototypowego systemu eksperckiego – przygotowanie i uruchomienie dialogowego systemu wspomagania projektowania struktur bez danych"*, wykonaną w Politechnice Poznańskiej (Wydział Elektryczny, Instytut Informatyki, Automatyki i Robotyki; promotor: dr inż. Tadeusz Morzy) oraz

Tomasz Szymacha za pracę pt. *„Równoległe algorytmy dla klas grafów definiowanych rekurencyjnie"*, wykonaną w Uniwersytecie Warszawskim (Wydział Matematyki, Informatyki i Mechaniki. Instytut Informatyki; promotor: doc. dr hab. Wojciech Rytter).

Trzecią nagrodę w wysokości 80 tys. zł otrzymuje

Witold Rudolf za pracę pt. *„JU-LEK – system autorski do tworzenia lekcji dla mikrokomputera szkolnego Elwro 800 Junior"*, wykonaną w Uniwersytecie Wrocławskim (Wydział Matematyczno-Fizyczno-Chemiczny, Instytut Informatyki; promotor: dr Ewa Gurbiel).

Przyznano ponadto trzy równorzędne **wyróżnienia** po 50 tys. zł; otrzymują je:

Piotr Imiela za pracę pt. *„Opracowanie i uruchomienie w języku C biblioteki procedur analizy obrazów pod kątem badania parametrów geometrycznych obszarów jednolitych"*, wykonaną w Politechnice Śląskiej w Gliwicach (Wydział Automatyki, Elektroniki i Informatyki, Instytut Informatyki; promotor: dr inż. Andrzej Kwiecień),

Piotr Dariusz Menzel i **Olgierd Unold** za pracę pt. *„Sieciowy system zarządzania bazą danych w dziekanacie"*, wykonaną w Politechnice Wrocławskiej (Wydział Elektroniki, Instytut Cybernetyki Technicznej; promotor: dr inż. Ryszard Klempous oraz

Przemysław Stpiczyński za pracę pt. *„Stogi i deki"*, wykonaną w Uniwersytecie Marii Curie-Skłodowskiej w Lublinie (Wydział Matematyki i Fizyki. Instytut Matematyki; promotor: doc. dr Światomir Ząbek).

Wręczenie nagród odbyło się w dniu 18 stycznia br. w ramach plenarnego posiedzenia Zarządu Towarzystwa.

Rzecznik prasowy
Polskiego Towarzystwa Informatycznego
BARBARA OSUCHOWSKA

ILONA BLUEMKE
Instytut Informatyki
Politechnika Warszawska

Architektury procesorów tablicowych tolerujących błędy

W 1986 roku w Informatyce zamieszczono cykl interesujących artykułów dotyczących systemów współbieżnych. Jako przykłady systemów współbieżnych przedstawiono procesy tablicowe stosowane do przetwarzania obrazów [9, 10]. Procesory tablicowe ze względu na swą regularną strukturę są doskonale dostosowane do realizacji w technologii VLSI. Przy wytwarzaniu układów VLSI znaczna ich część jest całkowicie niesprawna lub ma uszkodzone fragmenty już w procesie produkcji. Podczas pracy układu pewne jego elementy mogą także ulegać uszkodzeniom (uszkodzenia dynamiczne), a prawdopodobieństwo uszkodzenia jest tym większe, im większa jest złożoność elementu. Istotnym problemem jest takie zaprojektowanie architektury procesora tablicowego, by umożliwiał on tolerowanie uszkodzeń poprodukcyjnych i dynamicznych. Tolerowanie uszkodzeń odbywa się zazwyczaj przez rekonfigurowanie struktury procesora z ewentualnym włączeniem elementów zapasowych.

Poniżej omówiono ciekawsze rozwiązania procesorów tablicowych scalonych lub wykonanych z elementów dyskretnych tolerujących błędy:

- MPP,
  metoda Diogenesa,
  rozwiązanie Fortesa i Raghavendry (F–R),
- rozwiązanie Sami, Stefanelli i Negrini (S–S–N),
- CHIP.

W prezentowanych rozwiązaniach przyjęto założenie o wzajemnej niezależności błędów. Błędy połączeń są traktowane jako błędy elementów przetwarzających albo przyjmuje się założenie o niezawodności połączeń. Lokalizację błędu osiąga się stosując jednostki samodiagnozujące się jako elementy przetwarzające (S–S–N) lub wykorzystując testowanie z otoczenia układu. W niektórych z omawianych rozwiązaniach istnieje możliwość zmiany topologii połączeń (np. z kraty na drzewo binarne) i lepszego dostosowania do przetwarzanego algorytmu. Efektywność przetwarzania określonego algorytmu w systemie współbieżnym zależy od struktury połączeń. Inna jest optymalna ze względu na czas przetwarzania sieć połączeń, np. dla problemów sortowania, inna dla algorytmów macierzowych.

MPP

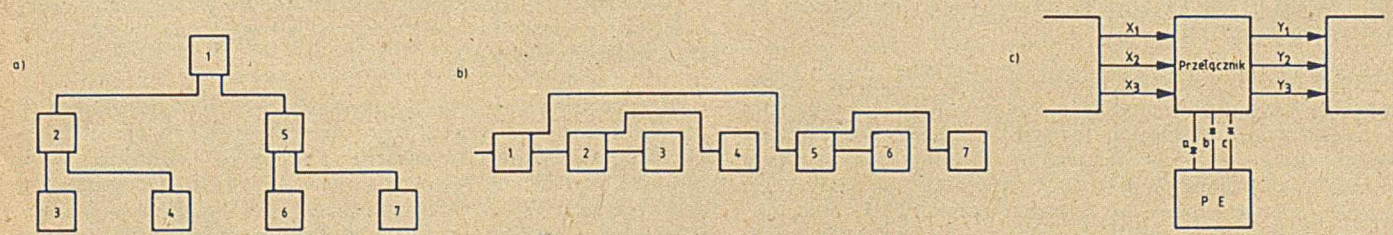
MPP (ang. *Massivelly Parallel Procesor*) [1, 2] jest procesorem tablicowym opracowanym i zrealizowanym w USA do przetwarzania obrazów satelitarnych. MPP składa się z tablicy procesorów o wymiarach 128 × 128 oraz z rezerwowego bloku procesorów 128 × 4. Wykonano go z układów zawierających po 4 × 2 procesory. W razie błędu nawet jednego tylko procesora, cały blok zawierający ten procesor jest izolowany, a zamiast niego jest włączany blok rezerwowy. Rozwiązanie to jest proste, łatwo realizowalne i wymaga niewielkiej liczby dodatkowych połączeń do przełączania bloku procesorów. Rekonfigurację tablicy procesorów można przeprowadzić bardzo szybko. MPP działa poprawnie, jeśli czas naprawy (wymiany) elementu uszkodzonego jest mniejszy niż czas między uszkodzeniami. W architekturze MPP udział elementów zapasowych w stosunku do liczby uszkodzonych elementów przetwarzających jest bardzo duży, ale złożoność sieci połączeń jest bardzo mała, a ponadto nie są wymagane żadne zmiany w algorytmie wykonywanym w procesie tablicowym. Procesor tablicowy MPP jest sterowany przez komputer PDP 11/45 [10].

METODA DIOGENESA

Bardziej złożoną sieć połączeń, umożliwiającą tworzenie wielu struktur połączeń, zastosowano w interesującym rozwiązaniu organizacji tablicy procesorów, zaproponowanym przez A.L. Rosenberg. Jest to tzw. metoda Diogenesa. Celem tej metody jest zrealizowanie dużej liczby alternatywnych topologii połączeń z użyciem liniowo rozmieszczonych elementów przetwarzających (PE), przełączników oraz połączeń tworzących „wiązki", kolejki lub stosy W metodzie tej przyjęto, że elementy przetwarzające są wytwarzane w postaci wektora i początkowo są zorganizowane liniowo. Taka topologia jest łatwo realizowana w technologii VLSI. Elementy nie są ze sobą połączone; połączenia między poszczególnymi elementami są realizowane przez przyłączenie do wiązki połączeń (lub kilku takich wiązek). Do każdego elementu przetwarzającego wchodzi pewna liczba połączeń z wiązki i pewna liczba linii wychodzących z elementu przetwarzającego jest doprowadzana do każdej z istniejących wiązek. Linie wchodzące i wychodzące z elementów przetwarzających są przyłączane do wiązek za pomocą przełączników ustawianych przez zewnętrzne linie sterujące. Przyjęto założenie, że połączenia i przełączniki są niezawodne.

Konfiguracja układu (utworzenie sieci połączeń) polega na sekwencyjnym przeglądaniu ciągu elementów przetwarzających i ustawianiu przełączników tak, by każdy element miał odpowiednią liczbę połączeń z wiązkami (dla procesora błędnego jest to zero połączeń). W metodzie zakłada się, że elementy przetwarzające były testowane (z otoczenia), a wyniki testowania są dostępne przez specjalną linię (GOOD). Wiązki połączeń mają strukturę szyny, by łatwo można było łączyć układy. Rekonfiguracja układu odbywa się przez ustawienie stanów przełączników każdego elementu przetwarzającego. Liczba wiązek połączeń i sposób ustawienia przełączników zależy od struktury sieci połączeń, która ma być utworzona. Taka organizacja umożliwia realizację wielu sieci połączeń, np. drzew, *n*–sześcianu, piramidy, macierzy. Dla struktur bardziej skomplikowanych niż wektor, wiązki połączeń muszą mieć organizację stosu (np. drzewo) lub kolejki (np. piramida) [3, 12].

Na rysunku 1a pokazano kompletne drzewo binarne o trzech poziomach, a na rysunku 1b jego linearyzację otrzymaną przejściem preorder. Na rysunku 1c przedstawiono strukturę pojedynczego elementu przetwarzającego (*PE*) z jedną linią wchodzącą i dwiema liniami wychodzącymi z elementu, z przełącznikiem oraz wiązką zawierającą trzy połączenia. Przełącznik łączy linie wchodzące i wychodzące z elementu przetwarzającego z wiązkami połączeń wg następujących reguł:



Rys. 1. Metoda Diogenesa: drzewo binarne o trzech poziomach (a), jego linearyzacja (b), struktura pojedynczego elementu (c)

1. PE – uszkodzony
X_1 i Y_1
X_2 i Y_2
X_3 i Y_3
2. PE jest liściem drzewa
X_1 i Y_2
X_2 i Y_3
X_3 i a
0 i Y_1
3. PE nie jest liściem drzewa
X_2 i Y_1
X_3 i a
b i Y_2
c i Y_3

Tak ustalone reguły połączeń umożliwiają utworzenie drzewa binarnego.

Błędne elementy przetwarzające mogą być zastępowane elementami nadmiarowymi (np. końcowymi elementami wektora) przez zmianę połączeń stosu, odłączenie elementu uszkodzonego i włączenie elementu nadmiernego, przy czym zostają zachowane połączenia (sąsiedztwa) wyłączonego błędnego elementu.

Metoda Diogenesa zaproponowana w 1983 roku nie doczekała się fizycznej realizacji, prawdopodobnie ze względu na trudności ze zrealizowaniem połączeń nie wnoszących opóźnień. Wadą tego rozwiązania jest znaczne wydłużenie połączeń przy liniowo rozmieszczonych błędach oraz zależność struktury układu od wymaganej sieci połączeń. Zaletą jest natomiast możliwość pełnego wykorzystania elementów przetwarzających oraz regularność struktury.

CHIP

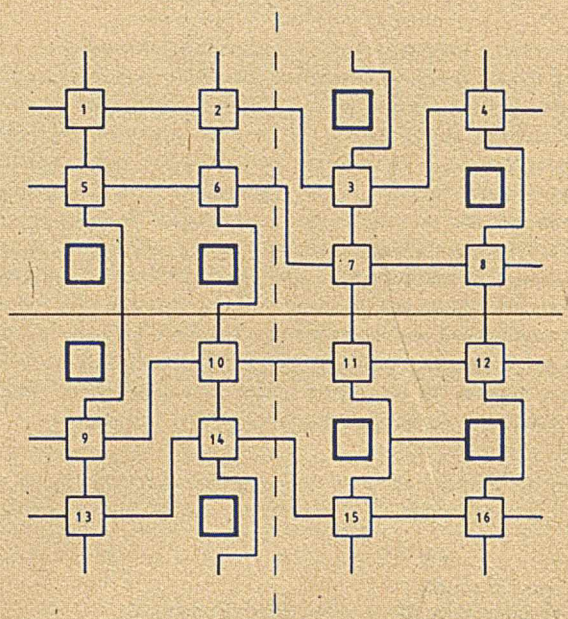
Interesującą architekturę zaproponował Snyder [15]. Jest to architektura komputera CHIP (ang. *Configurable Highly Parallel*) składającego się z trzech części:
- zbioru uniwersalnych identycznych procesorów,
- warstwy przełączników,
- sterownika.

Warstwa przełączników zawiera regularnie rozmieszczone programowalne przełączniki, połączone ze sobą ścieżkami. Jest ona najważniejszą częścią architektury i wyróżnia poszczególne typy komputerów CHIP. Procesory nie są ze sobą połączone bezpośrednio, lecz są dołączone w regularnych odstępach do warstwy przełączników. Procesory brzegowe są połączone z urządzeniami zewnętrznymi i z pamięcią. Każdy przełącznik ma lokalną pamięć do zapamiętania kilku swoich konfiguracji. Zmiana konfiguracji przełączników powoduje zmianę struktury sieci połączeń, np. z kraty na drzewo.

Sterownik jest odpowiedzialny za odpowiednie wypełnienie lokalnych pamięci przełączników. Wypełnienie pamięci przełączników odbywa się przed rozpoczęciem przetwarzania, przez specjalne połączenia; może się także odbywać równolegle z wprowadzeniem programów do pamięci procesorów. Każde miejsce lokalnej pamięci przełącznika służy jednemu typowi sieci połączeń i jest identyczne dla wszystkich przełączników (przykładowo, pozycja 1 odpowiada kracie, a pozycja 2 – drzewu binarnemu). Przetwarzanie w komputerze CHIP rozpoczyna sterownik takim ustawieniem stanu przełączników, by otrzymać wymaganą sieć połączeń (np. kratę). Następnie elementy przetwarzające synchronicznie wykonują instrukcje zapisane w ich pamięciach lokalnych. Poszczególne procesory nie mają informacji o sieci połączeń ani o swoich „sąsiadach"; wykonują operacje w rodzaju „*pobierz ze wschodu*", „*wyślij na północny-zachód*" itp. Nową fazę przetwarzania inicjuje sterownik wysyłając sygnał zmiany stanów przełączników. Równocześnie zmiana konfiguracji wszystkich przełączników powoduje zmianę sieci połączeń (np. na drzewo). W ten sposób CHIP może dostosowywać sieć połączeń do przetwarzanego algorytmu.

Architektura CHIP pozwala tolerować błędy. Tolerancję błędów można stosować statycznie, po zakończeniu procesu produkcji układu, oraz – z pewnymi ograniczeniami – dynamicznie podczas pracy. Jeśli po wyprodukowaniu układu zawierającego np. 10×10 procesorów okaże się, że koło 1/3 elementów jest uszkodzonych, to można na stałe ustawić stany pewnych przełączników tak, by błędne procesory były „omijane" i w ten sposób uzyskać układ zawierający np. 8×8 poprawnych procesorów.

Włączając do struktury elementy zapasowe można stosować rekonfigurację poprodukcyjną. Tablica elementów przetwarzających jest dzielona na podtablice (zawierające elementy zapasowe), wewnątrz których można wykonywać rekonfigurację. Na rysunku 2 przedstawiono cztery podtablice po 2×3 procesory. Każda z tych podtablic może się zrekonfigurować tolerując błędy dowolnych dwóch procesorów. Duża liczba elementów zapasowych nie gwarantuje jednak poprawnej pracy układu w razie powstania trzeciego błędu w dowolnej podtablicy, nawet przy dostępnych wolnych elementach zapasowych w innych podtablicach. Stosowanie dynamicznej tolerancji uszkodzeń procesorów w komputerze CHIP znacznie zwiększa złożoność sterowania zmianami stanów przełączników, a nie zapewnia pełnego wykorzystania elementów zapasowych.



Rys. 2. Struktura CHIP ze zrekonfigurowalnymi podtablicami (linią pogrubioną zaznaczono elementy uszkodzone)

Architektura CHIP jest dobrze dostosowana do technologii produkcji układów scalonych WSI. Prace nad wykonaniem CHIP w tej technologii rozpoczął Kye Hedlund w uniwersytecie w Purdue [6]. Do rekonfiguracji statycznej – poprodukcyjnej – zastosowano opisany wyżej algorytm ze zwiększoną liczbą elementów zapasowych. Podtablice liczą po 34 procesory, z czego wybiera się tylko po 22 poprawne. W celu zapewnienia odpowiedniej liczby poprawnych przełączników oraz elastycznej struktury połączeń, pozwalającej na „ominięcie" uszkodzonych procesorów, każdy przełącznik jest dublowany.

ARCHITEKTURA FORTESA-RAGHAVENDRY (F-R)

Procesor tablicowy zaproponowany przez Fortesa i Raghavendrę [4] zawiera, oprócz elementów przetwarzających połączonych w prostopadłą kratę (elementy brzegowe są połączone ze sobą połączeniami zewnętrznymi) i programowalnych przełączników, także dodatkową sieć połączeń w postaci linii biegnących równolegle do połączeń elementów przetwarzających (pionowo lub poziomo) w jednym lub obu kierunkach.

Wystąpienie błędu elementu przetwarzającego powoduje rekonfigurację tablicy procesorów, polegającą na wyeliminowaniu wiersza zawierającego uszkodzony element (SRE – ang. *succesive row elimination*) lub na alternatywnym wyeliminowaniu kolumny (ARCE – ang. *alternative row column elimination*). Usunięcie wiersza jest realizowane przez odpowiednie ustawienie stanu programowalnych przełączników, znajdujących się w bezpośrednim sąsiedztwie wiersza (kolumny) zawierającego uszkodzony element.

Wyłączenie wiersza (kolumny) elementów przetwarzających znacznie zmniejsza moc obliczeniową. Algorytm wykonywany w tablicy $n \times m$ i wykorzystujący wszystkie elementy musi być po wyłączeniu wiersza (lub kolumny) dzielony na „podalgorytmy" wykonywane w mniejszych podmacierzach. Wyniki otrzymane przez części algorytmu są przechowywane w pamięciach zewnętrznych i przekazywane do następnych części połączeniami zewnętrznymi.

Nie wszystkie algorytmy dostosowane do przetwarzania w procesorach tablicowych można wykonywać fragmentami, toteż nie wszystkie algorytmy tablicowe mogą być przetwarzane w architekturze F–R. Fortes i Raghavendra podają warunki, jakie musi spełniać algorytm, by mógł być wykonywany w zaproponowanej przez nich architekturze [4, 5].

Jeżeli podczas wykonywania algorytmu dane są przesuwane w kierunku niemalejących wartości jednej współrzędnej, to algorytm ten ma właściwość RR (rekonfigurowalność wzdłuż jednego kierunku), zwaną rekonfiguracją wierszami. Oznacza to, że do operacji wykonywanych w elemencie o współrzędnych *(i,j)* nie są potrzebne wyniki operacji otrzymanych w elemencie *(k,l)*, gdzie $k>i$. Obliczenia można podzielić na części i sekwencyjnie wykonywać w zrekonfigurowanej tablicy, przechowując wyniki częściowe w pamięci.

Jeżeli podczas wykonywania algorytmu przemieszczanie danych odbywa się w kierunku niemalejących wartości obu współrzędnych, to algorytm ten ma właściwość RCR (rekonfigurowalność wierszami lub kolumnami). Podział algorytmu może odbywać się w dwóch płaszczyznach. Fortes udowadnia [5], że **każdy algorytm tablicowy można przekształcić do równoważnej mu postaci mającej własność RR lub RCR.**

Procesor tablicowy F–R o n wierszach pozwala na tolerancję, w najgorszym razie n błędów rozmieszczonych w różnych wierszach. Powstanie każdego kolejnego uszkodzenia powoduje podział algorytmu na części, wykonywane kolejno w podtablicach, co znacznie zwiększa czas przetwarzania algorytmu. Stanowi to istotną wadę tej architektury. W opisanym powyżej procesorze tablicowym dodatkowa sieć połączeń konieczna do rekonfiguracji stanowi:
- 100% dla metody rekonfigurowania wierszami lub kolumnami (ARCE),
- 50% dla metody rekonfigurowania wierszami (SRE) podstawowej sieci połączeń.

Niekorzystną cechą tej architektury jest także konieczność ingerowania w przetwarzany algorytm, przekształcenie go do postaci mającej właściwość RR lub RCR i podział algorytmu na „części" mieszczące się w zrekonfigurowanym procesorze tablicowym. Zaletą jest natomiast prosty mechanizm eliminowania wiersza (kolumny) przez zmianę stanu przełączników.

ARCHITEKTURY NEGRINI-SAMI-STEFANELLI (S-S-N)

Sami, Stefanelli i Negrini zaproponowali metody rekonfigurowania prostokątnej tablicy identycznych elementów przetwarzających. Przyjęto, że procesor tablicowy składa się z elementów przetwarzających (podstawowych i zapasowych), sieci połączeń i sieci sterującej rekonfiguracją. Połączenia są wzdłuż prostopadłych osi, a przepływ informacji jest jednokierunkowy. Przyjęto także, że każdy element przetwarzający, identyfikowany przez współrzędne *i, j*, jest jednostką samotestującą, a wyniki samotestowania są przekazywane do połączonych elementów jako sygnały *e (i, j)*.

Element *(i, j)* jest po wyprodukowaniu błędnego wyniku wyłączany z dalszego funkcjonowania, a stan błędu jest zapamiętywany w przerzutniku *e (i, j)*. Zapamiętywanie stanu błędu powoduje, że błędy przemijające stają się błędami stałymi. Rekonfiguracja dynamiczna może odbywać się prawidłowo, gdy elementy przetwarzające nie mają pamięci lub zawierają tylko rejestry uaktualniane w każdym kroku przetwarzania. Wymaganie to nie dotyczy algorytmów rekonfiguracji stosowanych statycznie. Błędy połączeń są w tej architekturze traktowane jako błędy elementów przetwarzających. Może to uniemożliwiać dalszą pracę układu częściej niż w modelu, w którym zakłada się niezależne pojawianie się błędów połączeń i ich obsługę przez specjalne procedury: pozwala jednak na jednakowe traktowanie wszystkich rodzajów błędów i zredukowanie złożoności układów rekonfiguracji.

Negrini, Sami, Stefanelli zaproponowali dwie metody rekonfiguracji opisanej powyżej macierzy elementów przetwarzających:
- rekonfigurację bezpośrednią,
- rekonfigurację propagującą.

Rekonfiguracja bezpośrednia polega na wykonywanej natychmiast po zidentyfikowaniu błędnego elementu *(i, j)* rekonfiguracji wiersza *i* lub kolumny *j*. Rekonfiguracje są proste, aby liczba wymaganych dodatkowych połączeń nie była duża. Istnieją jednak pewne rozkłady błędów prowadzące do wystąpienia „fatalnego błędu", tj. sytuacji, w której rekonfiguracja nie jest możliwa mimo istnienia wolnych elementów zapasowych. Lepsze wykorzystanie elementów zapasowych otrzymano stosując metodę „podkradania" elementów, która powoduje propagację rekonfiguracji w tablicy. Układy sterujące rekonfiguracją są bardziej złożone, jednak liczba elementów zapasowych użyta do zapewnienia pracy układu po danej liczbie uszkodzeń jest mniejsza niż w rekonfiguracji bezpośredniej.

Struktura o bezpośredniej rekonfiguracji

Do tablicy $n \times n$ elementów przetwarzających dodano elementy zapasowe, w postaci dodatkowego wiersza i dodatkowej kolumny. Po powstaniu błędu następuje próba rekonfiguracji wykonywana przez globalną procedurę reorganizacji; odwzorowuje ona logiczne indeksy elementów (określające funkcje wykonywane w tablicy) na fizyczne adresy pracujących elementów. Takie globalne podejście jest bardziej złożone niż proste zastąpienie uszkodzonego elementu „jeden do jednego", pozwala jednak na małą liczbę dodatkowych połączeń i ogranicza dodatkowe opóźnienia.

W celu zagwarantowania bezpośrednich połączeń między „logicznie" przyległymi elementami, każdy element przetwarzający jest wyposażony w układy wyznaczania logicznych adresów (swoich i elementów przyległych) i uaktywniania odpowiednich połączeń. Układy takie są identyczne dla każdego elementu, regularność macierzy jest zachowana.

Rekonfiguracja odbywa się wzdłuż dwóch osi, w kierunku przepływu informacji do elementów zapasowych. Mając błędny element najpierw należy wybrać kierunek rekonfiguracji:
- **pionowa** – powoduje pobranie elementu zapasowego z dodatkowego wiersza tablicy,
- **pozioma** – powoduje pobranie elementu z dodatkowej kolumny.

Po wykryciu nowego błędu **cała macierz** podlega rekonfiguracji, a więc poprzednie przypisania elementów zapasowych mogą być zmodyfikowane. Rekonfiguracja przebiega dwustopniowo:
– określenie kierunku rekonfiguracji,
– odwzorowanie logicznych współrzędnych na fizyczne adresy pracujących elementów.

Bezpośrednią konsekwencją powyższych zasad rekonfiguracji są ograniczenia rozkładu błędów, dla których można stosować rekonfigurację. Prowadzi to do następujących reguł:
– w każdym wierszu tylko jeden błąd może powodować rekonfigurację wzdłuż osi poziomej,
– w każdej kolumnie maksymalnie jeden błąd może powodować rekonfigurację wzdłuż osi pionowej.

Kolejność stosowania rekonfiguracji pionowej i poziomej ma wpływ na końcową konfigurację tablicy. Przyjęto, że najpierw wykonuje się rekonfigurację poziomą.

Na rysunku 3 przedstawiono działanie algorytmu w tablicy zawierającej 5×5 elementów przetwarzających oraz wiersz i kolumnę elementów zapasowych, przy czym najpierw wykonywana jest rekonfiguracja pozioma a następnie pionowa. Proponowaną strukturę można rozszerzyć o większą liczbę dodatkowych elementów zapasowych, np. o 2 wiersze i 2 kolumny. Zwiększy się wówczas prawdopodobieństwo pracy układu po wystąpieniu danej liczby losowo rozmieszczonych uszkodzeń, lecz zwiększy się także złożoność układu i obszar potrzebny na jego wykonanie. Badania symulacyjne struktury o bezpośredniej rekonfiguracji wykazały [13], że prawdopodobieństwo pracy układu równe 0,5 można zagwarantować tylko dla liczby błędów nie przekraczającej 30% liczby elementów zapasowych. Autorzy zaproponowali rozszerzenie tej metody dające lepsze wykorzystanie elementów zapasowych – struktury o propagującej rekonfiguracji.

W strukturach tych zastosowano rozszerzony algorytm konfiguracji, polegający na poszukiwaniu elementów zapasowych nie tylko w wierszu lub kolumnie, w której wystąpił błąd, lecz także w wierszach przyległych. Sami i Stefanelli [13] prezentują dwa algorytmy rekonfiguracji macierzy zawierającej jedną kolumnę elementów zapasowych:
– algorytm „stałego podkradania" (ang. *fixed stealing*),
– algorytm „zmiennego podkradania".

W każdym z tych algorytmów pojedynczy element może znajdować się w jednym z trzech stanów:
1) element poprawny – jego logiczny numer wiersza odpowiada numerowi (adresowi) fizycznemu,

2) element uszkodzony,
3) element został „pobrany" do rekonfiguracji do wiersza o większym numerze; logiczny i fizyczny numer wiersza są różne (numer logiczny jest zawsze mniejszy niż fizyczny).

Algorytm „stałego podkradania"

W strukturze elementów przetwarzających dodano jedną kolumnę elementów zapasowych (z prawej strony). Niech w wierszu i będą dwa błędne elementy: (i, k) oraz (i, h) przy czy $k<h$, a wiersz $i-1$ nie zawiera elementów uszkodzonych. Element (i, k) jest elementem błędnym, znajdującym się najbardziej z lewej strony i powoduje rekonfigurację w stronę prawą, zmieniając logiczne indeksy elementów (i, j) dla $j >= k$ na $(i, j-1)$. Element (i, h) „kradnie" pozycję elementu $(i-1, h)$, a „skradziony" element z kolei zachowuje się tak, jak gdyby był elementem błędnym, znajdującym się najbardziej z lewej strony w wierszu $i-1$ i rekonfiguruje wiersz $i-1$ w stronę prawą. Obecność n błędów w wierszu i powoduje pobranie elementów zapasowych w wierszu $i-1$, $i-2$,... $i-n$. Warunkiem koniecznym rekonfiguracji jest $i >= n$.

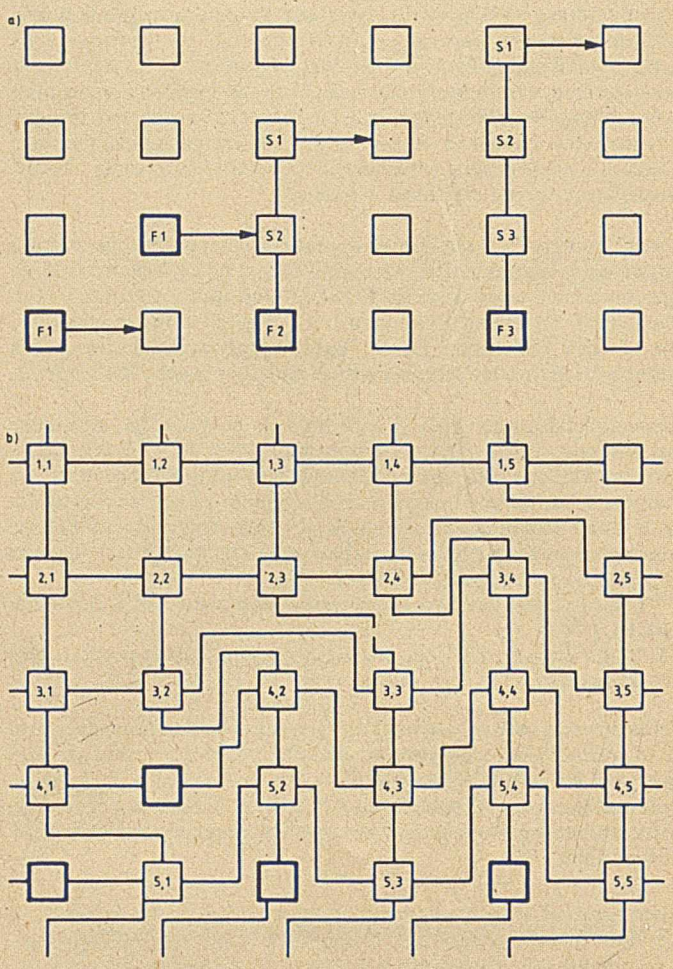
Poniższy przykład ilustruje działanie tego algorytmu (rys. 4). Przeglądanie macierzy odbywa się począwszy od ostatniego wiersza i pierwszej kolumny ($i=5$, $j=1$). W wierszu $i = 5$ są trzy elementy uszkodzone. Element znajdujący się najbardziej z lewej strony ($F1$) inicjuje rekonfigurację macierzy pobierając element zapasowy z wiersza 5, a elementy uszkodzone $F2 = (5,3)$ i $F3 = (5,5)$ „kradną" elementy z wiersza $i = 4$ $S2 = (4,3)$ i $S3 = (4,5)$. Procedura jest wykonywana rekurencyjnie dla zmniejszonej wartości i aż do osiągnięcia wiersza początkowego ($i=1$). Na rysunku 4b pokazano logiczne indeksy i połączenia elementów po zakończeniu rekonfiguracji macierzy. Rekonfiguracja nie jest możliwa, gdy szukając elementu do pobrania w wierszu poprzednim, trafia się na element uszkodzony lub gdy znaleziony element też dąży do pobrania elementu. Sami, Stefanelli i Negrini podają schemat układu sterującego opisanym powyżej algorytmem rekonfiguracji [13]. Metoda ta powoduje deformację pionową i poziomą logicznych indeksów elementów.

Istotną wadą tego algorytmu jest brak możliwości rekonfiguracji wówczas, gdy w wierszu początkowym jest więcej niż jeden element błędny lub „skradziony". Proste rozwiązanie tego problemu polega na dodaniu zapasowego wiersza poprzedzającego początkowy wiersz macierzy. Prawdopodobieństwo pracy układu po n błędach jest podobne, jak w algorytmie rekonfiguracji bezpośredniej, zbliżona jest także złożoność układu sterującego rekonfiguracją.

Algorytm „zmiennego podkradania"

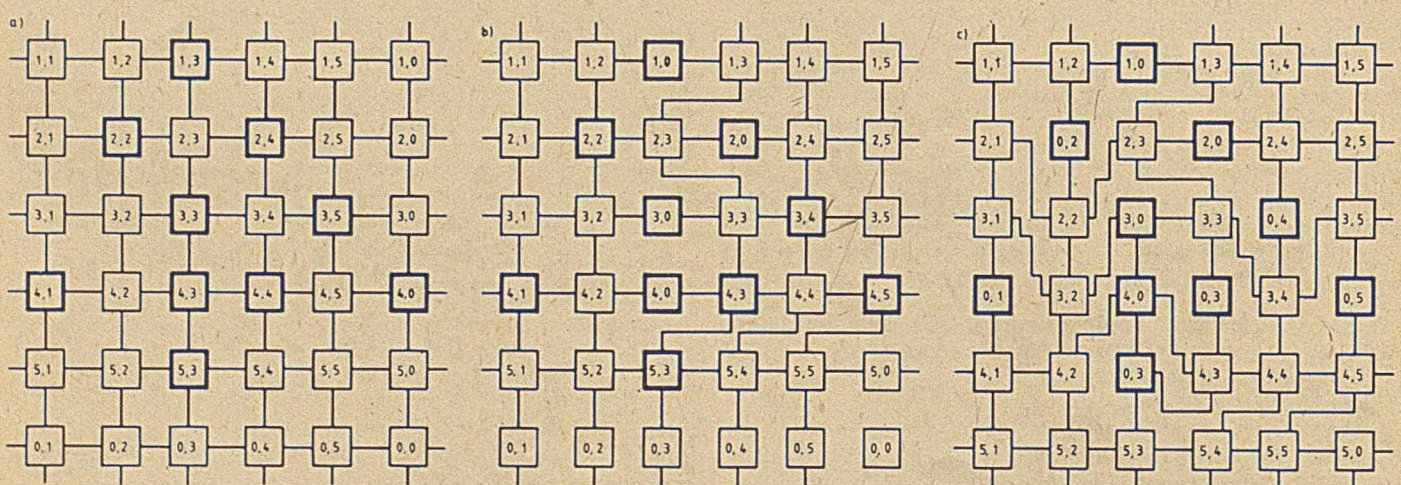
W algorytmie „stałego podkradania" korzystano ze sztywnego założenia, że element pobierany w miejsce uszkodzonego jest elementem bezbłędnym. W algorytmie „zmiennego podkradania" zrezygnowano z tego założenia, zwiększając jednak złożoność układu sterującego rekonfiguracją.

Aby zredukować prawdopodobieństwo powstania fatalnego błędu elementy błędne i „kradnące" $(i-j)$ w wierszu i bada się w celu sprawdzenia, czy w wierszu $i-1$ element $(i-1, j)$ jest błędny. Jeśli tak, to element (i, j) rekonfiguruje w wierszu i w prawo, a pozostałe elementy pobierają element z wiersza $i-1$. Jeżeli w wierszu $i-1$ nie ma elementu błędnego w kolumnie j, a (i, j) jest elementem błędnym lub „kradnącym", to jest stosowany algorytm „stałego podkradania".



Rys. 4. Algorytm stałego podkradania: rozkład błędów (a) i tablica po rekonfiguracji (b)

Rekonfiguracja nie jest możliwa, gdy w wierszu początkowym ($i=1$) więcej niż jeden element jest uszkodzony lub „ukradziony". Rekonfiguracja nie jest także możliwa, gdy więcej niż jeden element błędny lub „kradnący" w wierszu i znajduje element uszkodzony w tej samej kolumnie w wierszu $i-1$. Złożoność układu sterującego rekonfiguracją jest znacznie większa (około dwukrotnie) niż w algorytmie „stałego podkradania", a prawdopodobieństwo układu pracy po n błędach tylko nieznacznie większe.



Rys. 3. Rekonfiguracja bezpośrednia: rozkład błędów (a), rekonfiguracja pozioma (b), tablica po rekonfiguracji (c)

Rozszerzenie architektury

Przedstawione algorytmy dotyczyły tablicy prostokątnej o jednokierunkowym przepływie informacji. Rozszerzenia na inne regularne struktury, np. na tablicę prostokątną o dwukierunkowym przepływie informacji lub macierze heksagonalne, są proste. Do każdego wejścia do elementu przetwarzającego należy dodać multiplekser określający, który sygnał między zbiorem alternatywnych sygnałów ma być skierowany do elementu.

Celem zaproponowanego rozwiązania jest osiągnięcie kompromisu między wykorzystaniem elementów zapasowych a wymaganymi dodatkowymi połączeniami. Przedstawione algorytmy rekonfiguracji dla pewnych rozkładów błędów uniemożliwiają rekonfigurację mimo istniejących wolnych elementów zapasowych, ale zapewniają małą złożoność sieci połączeń i układów sterujących rekonfiguracją. Drugą podstawową zaletą proponowanego rozwiązania jest zachowanie regularności układu.

PORÓWNANIE PROCESÓW TABLICOWYCH

Porównanie procesów tablicowych można przeprowadzać, na przykład, w zależności od:
- prawdopodobieństwa pracy układu po określonej liczbie błędów,
- dodatkowego czasu przetwarzania, potrzebnego do przeprowadzenia rekonfiguracji,
- dodatkowych układów sprzętowych, ewentualnie dodatkowej powierzchni półprzewodnika, dla procesorów tablicowych wykonanych w technologii scalonej.

Dodatkowa powierzchnia półprzewodnika jest potrzebna przy zastosowaniu jako elementów przetwarzających jednostek samotestowalnych (rozwiązanie S-S-N), do umieszczenia elementów zapasowych (np. CHIP, S-S-N) oraz do prowadzenia dodatkowych połączeń (rozwiązanie F-R). Zwiększenie powierzchni półprzewodnika może być także spowodowane umieszczeniem wewnątrz układu VLSI sieci sterującej rekonfiguracją (S-S-N). Czas potrzebny do wykonania rekonfiguracji układu jest pomijalny w MPP, a może stać się znaczący przy bardziej złożonych metodach rekonfiguracji, np. w rozwiązaniu F-R.

Zdolność tolerancji określonej liczby błędów zależy od liczby i stopnia wykorzystania elementów zapasowych oraz „inteligencji" algorytmu rekonfiguracji. Zależność między rozważanymi wielkościami są niezwykle skomplikowane i praktycznie nie ma możliwości obiektywnego porównania jakości procesorów tablicowych tolerujących błędy. Niezwykle trudne jest także wyznaczenie porównawczej niezawodności struktur z rekonfiguracją i struktur bez rekonfiguracji. Różnice w niezawodności zależą nie tylko od stosowanego algorytmu niezawodności i niezawodności elementu, lecz także od zwiększenia względnej powierzchni układu i ogólnej złożoności.

W architekturze procesora tablicowego tolerującego błędy powinna być zachowana równowaga między wieloma czynnikami, np. złożonością algorytmu rekonfiguracji, stopniem wykorzystania elementów poprawnych, lokalizacją uszkodzeń, dodatkowymi układami (sieciami połączeń, sieciami sterującymi).

LITERATURA

[1] Batcher K.: Design of a massivelly parallel processor. IEEE Trans. on Computers, Vol. C-29, No. 9, pp. 836-840, 1980
[2] Batcher K.: Bit-serial parallel processing systems. IEEE Trans. on Computers, Vol. C-31, No. 5, pp. 377-384, 1982
[3] Chung R.K., Leighton F.T., Rosenberg A.L.: Diogenes – a methodology for designing fault-tolerant VLSI processor arrays. FTCS 13, pp. 26-32, 1983
[4] Fortes J.A.B., Raghavendra C.S.: Gracefully degradable processor array. IEEE Trans. on Computers, No. 11, pp. 1033-1044, 1985
[5] Fortes J.A.B., Moldovan D.I.: Partitioning of algorithms for execution in fixed size VLSI architektures. Technical Report PPP75, Dept. Electrical Engineering, Universites of Southern California, 1983
[6] Heldund K.: VLSI Design, July-August 1983
[7] Iszkowski W.: Architektury systemów n-mikroprocesorowych. Informatyka, nr 7-8, 1986
[8] Negrini R., Sami M., Stefanelli R.: VLS-WSI arrays a fault tolerant approach (niepublikowane)
[9] Owczarczyk J., Stolarski M., Woźniak E.: Architektura współbieżnych systemów przetwarzania obrazów. Informatyka, nr 7-8, 1986
[10] Owczarczyk J., Stolarski M., Woźniak E.: Procesory tablicowe – współbieżne systemy przetwarzania obrazów. Informatyka, nr 11-12, 1986
[11] Potter J.L.: Image processing on the massivelly parallel processor. Computer, No.1, pp. 62-67, 1983
[12] Rosenberg A.L.: The Diogenes approach to testable fault-tolerant VLSI processor array. IEEE Trans. on Computers, Vol. C-32, No. 10, pp. 902-910, 1983
[13] Sami M., Stefanelli R.: Reconfigurable architektures for VLSI processing arrays. Proc. NCC 83, AFIPS
[14] Sami M. Stefanelli R.: Fault – tolerance of VLSI processing arrays – the time redundancy approach. Proc. 1984 Real-Time Systems Symp.
[15] Snyder L.: Introduction to the configurable, highly parallel computer. Computer, No. 1, 1982
[16] Stefanelli R.: Multiple error correction in arithmetic units with triple redundancy. Proc. Euromicro, pp. 205-216, 1984.

Książki z dziedziny sztucznej inteligencji rozprowadzane przez firmę J. Wiley

A. Thayse (ed.): From Standard Logic to Logic Programming. Introducing a Logic Based Approach to Artificial Intelligence. J. Wiley, lipiec 1988, s. 360

Książka dostarcza materiału do studiów nad sztuczną inteligencją z punktu widzenia wykorzystywanej w niej logiki: od najbardziej teoretycznych aspektów logiki do technik wykorzystywanych w praktyce programowania logicznego. Po wprowadzeniu podstaw logiki pierwszego rzędu pokazuje, jak zapis i rachunek logiczny mogą być wykorzystane do reprezentowania wiedzy oraz do wnioskowania. Przedstawia reguły przejścia od reprezentacji logicznej do reprezentacji graficznej i obiektowej. Następnie prezentuje logiki modalne, niemonotoniczne, domyślne i samopoznawcze (ang. *autoepistemic*). Po wprowadzeniu gramatyk pokazuje, jak logika i gramatyki prowadzą do takich języków programowania, jak Prolog.

S. Savory (ed.): Artificial Intelligence and Expert Systems. Ellis Horwood, kwiecień 1988, s. 278

Interesująca próba zestawienia akademickich badań w dziedzinie sztucznej inteligencji z rzeczywistymi potrzebami i wymaganiami przemysłu i gospodarki, na przykładzie prac nad zastosowaniami przemysłowymi podjętymi w zachodnioniemieckiej firmie Nixdorf Computer AG. Książka zawiera m.in. ogólny przegląd dotyczący stanu prac w dziedzinie sztucznej inteligencji, opis postulowanej architektury systemu ekspertowego, a także wprowadzenie do Prologu, przegląd uczenia maszynowego, inżynierii wiedzy i uczenia przez podawania przykładów.

M. L. Barrett, A. C. Beerel: Expert Systems in Business. A. Practical Approach. Ellis Horwood, kwiecień 1988, s. 260

Autorzy są praktykami w dziedzinie budowy i wykorzystania systemów ekspertowych. W książce koncentrują się na wyjaśnieniu – w jasny i często dość szczegółowy sposób – metod i technik budowy tych systemów, przekazując sporo z własnego doświadczenia.

B. Whitby: Artificial Intelligence. A Handbook of Professionalism. Ellis Horwood, czerwiec 1988, s. 194

Książka zawiera socjologiczne rozważania o naturze sztucznej inteligencji, jej wpływie na społeczeństwo i rodzaju odpowiedzialności, która jest związana z pracami w tej dziedzinie.

M. K. Chytil, R. Engelbrecht (eds.): Medical Expert Systems. Sigma Press, kwiecień 1988, s. 205

Przegląd medycznych systemów ekspertowych pracujących w komputerach osobistych. Rozważa się problemy związane z wprowadzeniem do medycyny techniki mikrokomputerowej oraz wpływ, jaki wywarło hasło piątej generacji na tę dziedzinę. Podkreśla się rolę teoretycznych narzędzi, istotnych w projektowaniu systemów eksportowych.

E. Rietman: Experiments in Artificial Neural Networks. TAB Books Inc., maj 1988, s. 160

Opis możliwości symulowania systemów nerwowych mózgu i zastosowania tej symulacji w przetwarzaniu informacji. Po wprowadzeniu pojęć związanych z rozproszonym przetwarzaniem równoległym i symulacją cyfrową, dochodzi się do takich określeń, jak „elektroniczne sieci nerwowe" i „obliczenia nerwowe" (ang. *neural computing*) oraz opisów czterech eksperymentów.

M. Greenwell: Knowledge Engineering for Expert Systems. Ellis Horwood, lipiec 1988, s. 180

Autor skupia uwagę na uzyskiwaniu wiedzy od eksperta. Stara się opisać najlepsze metody przeprowadzania rozmów i analizy uzyskiwanych informacji. Proponuje do tych celów podejście strukturalne. Podaje przykłady ze swojej praktyki z systemem EMEX.

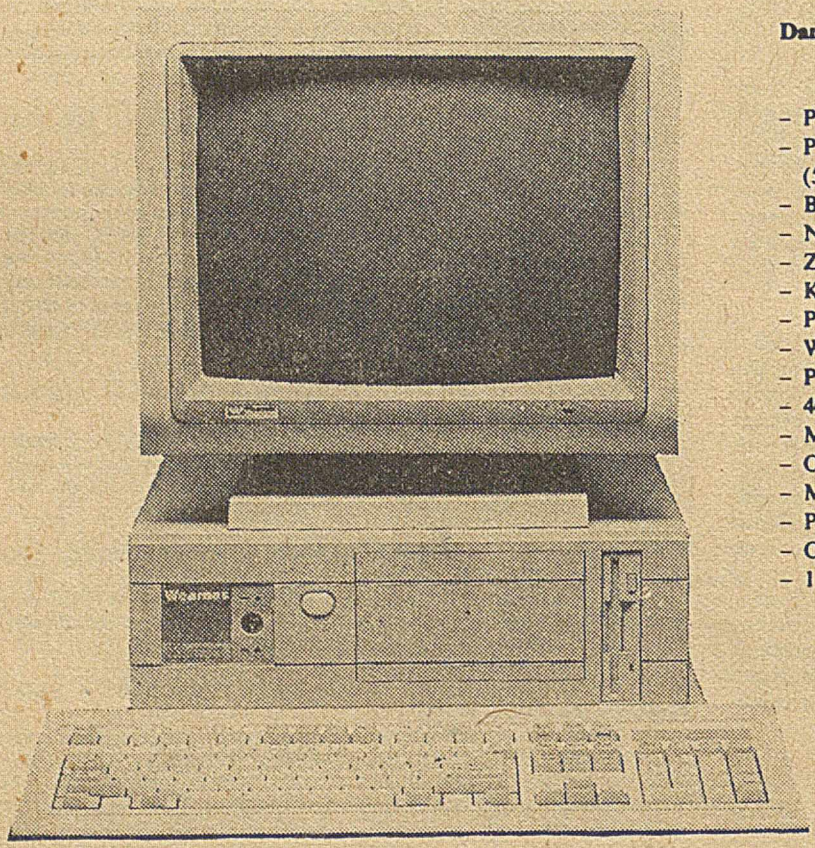
dokończenie na s. 12

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 – 12.5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

BOLDLINE COMPUTERS
GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40–045 Katowice
telefon i telefaks: 518-628
teleks: 315687 COMP PL

DYSTRYBUTORZY:
PTH „TECHMEX"
43–300 Bielsko-Biała
ul. M. Curie Skłodowskiej 13
tel.: 42-198, 47-555, telefaks: 47-624,
teleks: 25325

EO/1123/89

SŁAWOMIR CZEPIELEWSKI
ZDZISŁAW NOWAKOWSKI
Instytut Podstaw Informatyki PAN
Warszawa

Środowisko programowe do projektowania pakietów elektronicznych systemów mikroprocesorowych

Pakiety sprzętowe stosowane w systemach mikroprocesorowych są skomplikowanymi urządzeniami elektronicznymi. Na ogół znajduje się w nich bardzo wiele wzajemnie połączonych układów scalonych, gęsto upakowanych ze względu na dążenie do miniaturyzacji i zwiększenia szybkości. W rezultacie zadanie sprawnego i bezgłębnego zaprojektowania pakietu zaczęło przerastać możliwości projektanta, który nie może już opierać się tylko na doświadczeniu i intuicji. Powstała więc konieczność stworzenia odpowiednich narzędzi wspomagających pracę projektanta.

W ostatnich latach stworzono wiele programów narzędziowych do projektowania pakietów zarówno dla dużych komputerów, jak i dla mikrokomputerów. Najbardziej popularne, przeznaczone dla komputerów zgodnych z IBM PC i pracujących pod kontrolą systemu operacyjnego DOS, to: P–CAD, Questrac, Racal–Redac i Orcad.

Większość współczesnych programów wspomagających projektowanie pakietów sprzętowych można podzielić na cztery podstawowe moduły:

- edytor schematów,
- symulator,
- moduł projektujący obwód drukowany,
- moduł przygotowujący dokumentację.

Pewne programy mogą zawierać tylko niektóre moduły, inne przeznaczone do specjalnych zastosowań – mogą być wyposażone w dodatkowe moduły.

REDAGOWANIE SCHEMATÓW

Każdy program wspomagający musi zawierać edytor schematów, umożliwiający wprowadzenie schematu układu cyfrowego do pamięci komputera. Schemat układu można wczytywać trzema sposobami, jako:

- odwzorowanie schematu technicznego,
- wprowadzenie listy elementów i połączeń wraz z typami obudów,
- wprowadzenie schematu przy użyciu dyskretyzatora.

Pierwszy sposób polega na wybieraniu przez projektanta symboli właściwych układów cyfrowych z załączonej biblioteki programów i umieszczaniu ich w odpowiednim miejscu na ekranie monitora. Projektant może obracać wybrane symbole układów o 90°. Następnie, korzystając z myszy, klawiatury lub dyskretyzatora, projektant może poprowadzić linie łączące ze sobą końcówki symboli poszczególnych układów cyfrowych. Programy wspomagające projektowanie zawierają w swoich bibliotekach symboli układów elektronicznych często po kilka tysięcy elementów. Poza tym na ogół można samodzielnie definiować symbole układów elektronicznych i umieszczać je w bibliotece. Opcja ta jest istotna w dobie ciągłego wprowadzania na rynek nowych układów, gdyż umożliwia stałe aktualizowanie bibliotek.

Drugi sposób polega na złożeniu schematu układu cyfrowego z elementów (prostszych składowych). W wyniku kodowania otrzymuje się listę elementów składowych schematu, ze zbiorem elementów połączonych z wejściem i wyjściem każdego z nich. Przy dużych pakietach takie kodowanie jest bardzo pracochłonne i łatwo popełnić trudne do wykrycia błędy. Otrzymany w jego wyniku zbiór jest wpisywany przy użyciu klawiatury. Istotną wadą tej metody jest brak możliwości wyprowadzenia schematu układu cyfrowego na urządzenia wyjściowe komputera.

Trzeci sposób polega na wczytaniu schematu układu przy użyciu dyskretyzatora. Projektant umieszcza wczytywany schemat na dyskretyzatorze i kopiuje go.

Schemat wprowadzony do pamięci komputera za pomocą jednej z opisanych metod można już dość łatwo zmieniać oraz drukować jego poprawione wersje. Unika się w ten sposób bardzo pracochłonnego wielokrotnego przerysowywania, uciążliwego zwłaszcza na etapie testowania i uruchamiania sprzętu.

Edytor schematów umożliwia kontrolowanie poprawności schematu ze względu na zachowanie podstawowych praw elektryczności i reguł projektowania. W ten sposób można wykryć wszelkie zwarcia, brak zasilania lub uziemienia, nie przyłączone końcówki itp.

SYMULACJA UKŁADU

Symulator jest konwersacyjnym programem do symulowania działania układów cyfrowych umożliwiającym zweryfikowanie poprawności logicznej zaprojektowanego i wczytanego schematu. Poszczególne bloki schematu mogą być opisywane za pomocą bramek logicznych lub przez definicję funkcjonalną układu modelowego, tworzoną przy użyciu specjalnych języków. Projektant może tworzyć hierarchiczny projekt realizowany z bramek i typowych układów scalonych. Tak opisany schemat pozwala obserwować przebiegi czasowe w dowolnych punktach układu, kontrolować stany niestabilne w wybranym momencie, definiować wyjściowe sygnały sterujące itp. Wyniki symulacji można zapisywać w postaci przebiegów czasowych lub w postaci tablic prawdy. Wyniki otrzymywane za pomocą symulatora mają szczególne znaczenie w procesie testowania i uruchamiania pakietów, gdyż umożliwiają porównanie rzeczywistych przebiegów czasowych z przebiegami modelowymi, co w razie niezgodności pozwala dość łatwo zlokalizować miejsca uszkodzeń rzeczywistego pakietu.

PROJEKTOWANIE OBWODU DRUKOWANEGO

Schemat pakietu, po wczytaniu, jest przez edytora przekodowywany i zapisany w postaci zbioru danych wejściowych dla modułu projektującego obwód drukowany. Przed uruchomieniem tego modułu określa się rozmiary płyty, na której ma być wykonany montaż. Na ogół jest to standardowa tzw. eurokarta lub jej wielokrotności. Następnie wczytuje się odpowiednie pliki wyjściowe edytora schematów. Projektant otrzymuje na ekranie zbiór układów tworzących pakiet. Układy te należy rozmieścić na płytce – ręcznie lub automatycznie. Należy zauważyć, że bardzo doświadczony projektant jest w stanie znacznie lepiej rozmieścić elementy niż dostępne obecnie na rynku programy, choć w znacznie dłuższym czasie. Projektantom mniej doświadczonym zaleca się stosowanie programów automatycznych. Projektant zawsze może skorygować zaprojektowane przez program rozmieszczenie elementów.

Drugim krokiem w projektowaniu obwodu drukowanego jest prowadzenie ścieżek, które również może być realizowane ręcznie lub automatycznie. Przy złożonych pakietach prowadzenie automatyczne na ogół nie pozwala wyznaczyć wszystkich ścieżek i stosuje się je w celu zweryfikowania właściwego umiejscowienia elementów. Gdy procentowy udział ścieżek prowadzonych automatycznie jest mały, należy zweryfikować rozmieszczenie elementów, szukając takiego ich ułożenia, aby wszystkie lub prawie wszystkie ścieżki poprowadzić automatycznie. Projektant może w każdej chwili korygować bądź zmieniać sposób prowadzenia ścieżek, przechodząc na tryb pracy ręcznej.

Podstawową funkcją celu optymalizacji prowadzenia ścieżek jest minimalizacja całkowitej długości wszystkich ścieżek obwodu drukowanego. W trakcie prowadzenia ścieżek można określić takie parametry, jak maksymalny czas realizacji pojedynczej ścieżki, maksymalną liczbę przelotek dla jednego połączenia, maksymalne odejście ścieżek w pionie i w poziomie od prostokątnego ułożenia, grubość ścieżek itp. Istniejące programy umożliwiają projektowanie druków wielowarstwowych liczących nawet kilkadziesiąt warstw.

WYKONANIE PAKIETU I DOKUMENTACJI

Danymi wyjściowymi modułu projektującgo są pliki sterujące fotoplotery i perforatory taśm do wiertarek sterowanych numerycznie. Na ich podstawie fotoploter wytwarza maskę obwodu drukowanego, umożliwiającą przeniesienie schematu połączeń na nośnik rzeczywistego pakietu. Taśma perforowana umożliwia automatyczne wykonanie otworów na nośniku pakietu przy użyciu odpowiedniej wiertarki sterowanej numerycznie. Na tak przygotowanym nośniku należy umieścić układy elektroniczne zgodnie ze schematem i po zlutowaniu przekazać do testowania, którego pozytywny wynik kończy proces tworzenia pakietu.

Poważnym problemem jest przygotowanie dokumentacji technicznej wykonanego pakietu. Wykonywanie jej metodami tradycyjnymi zajmuje bardzo dużo czasu i łatwo, ze względu na ciągłe zmiany i poprawki, o popełnienie błędów. W celu automatycznego generowania dokumentacji stworzono specjalne moduły programu projektującego. Umożliwiają one, na podstawie danych wyjściowych innych modułów, wykonanie niezbędnych rysunków i zestawień. Ostatnio wprowadzone na rynek pakiety programowe umożliwiają nawet otrzymanie trójwymiarowych rysunków pakietów oraz ich dowolnych przekrojów.

IMPLEMENTACJA PROGRAMÓW ORCAD I RACAL–REDAC

W celu przygotowania środowiska programowego wspomagającego projektowanie pakietów elektronicznych, w laboratorium Architektury Systemów Komputerowych Instytutu Podstaw Informatyki PAN zainstalowano, uruchomiono i przetestowano pakiet Orcad Racal–Redac. Pakiety te umieszczono na dysku stałym komputera IBM PC, AT ze sterownikiem graficznym karty i monitorem EGA. Rozdzielczość i liczba kolorów standardu EGA okazała się wystarczająca do czytelnej wizualizacji wyników projektowania, w przeciwieństwie do testowanego początkowo sterownika graficznego według standardu CCA.

Orcad ze względu na swoje walory użytkowe jest bardzo popularnym programem do redagowania schematów. Jego podstawowy plik o nazwie Draft jest dogodnym i rozwiniętym funkcjonalnie edytorem. Biblioteka zawiera ponad 3000 elementów, można pamiętać do 100 makroprogramów, rozmiary wprowadzanego schematu są ograniczone tylko rozmiarem pamięci komputera. Orcad zawiera wiele innych programów:

- Composer, umożliwiający wprowadzenie do biblioteki zdefiniowanych przez projektanta układów elektronicznych,
- Partlist, drukujący listy wszystkich elementów wraz z ich parametrami, co jest pomocne przy sporządzaniu dokumentacji,
- Ercheck, sprawdzający zgodność wprowadzonego schematu z podstawowymi zasadami projektowania,
- Netlist, umożliwiający generowanie danych wyjściowych edytora w formatach akceptowanych przez wiele pakietów do projektowania obwodów, jak: Apliconbravo, Algorex, Caley, Cadnetix, Computervision, Edif, P–Cad, Spice, Wirelist Racal–Redac itp.

Pakiet Racal–Redac składa się z czterech podstawowych grup programów:

- Redlog, o zakresie funkcjonalnym Orcada, ma jednak znacznie mniej opcji, dlatego dogodniejsze jest stosowanie Orcada, zwłaszcza że generuje on pliki wyjściowe w formacie Racal–Redac;
- Radsim jest pakietem programów do symulacji logicznej i analizy czasowej; symulator układu logicznego wykrywa konflikty na szynie i impulsy pasożytnicze; analizator czasowy uwzględnia dyspersję impulsów cyfrowych; wykrywa punkty niejednoznaczne, pozwala uniknąć redundancji i zlokalizować węzły blokujące; jako opcję można dołączyć symulator błędów pomagający w przygotowaniu testów: maksymalna liczba bramek, wynosi 1000 dla komputera PC/XT i 3000 dla PC/AT; biblioteka układów elektronicznych zawiera 2500 elementów;
- Redboard jest modułem do projektowania obwodów drukowanych wielowarstwowych; maksymalna liczba warstw wynosi 16, maksymalna liczba końcówek na element – 256; liczba możliwych szerokości ścieżek jest ograniczona do 8, na jednej karcie można umieścić do 511 elementów i do 1900 ścieżek; pakiet ten zawiera dwa programy do automatycznego rozmieszczania elementów, trzy różne programy do prowadzenia ścieżek oraz program do automatycznej kontroli schematu; pozwala także generować pliki wyjściowe na fotoploter i perforator taśmy do sterowania drukarek numerycznych;
- Red–Draw pozwala otrzymać dwuwymiarowe rysunki pakietów elektronicznych w dowolnych przekrojach i rzutach.

Zastosowanie programów wspomagających projektowanie pakietów elektronicznych umożliwia znaczne zmniejszenie kosztów oraz przyspieszenie prac przy ich projektowaniu i budowie, zwłaszcza w przypadku skomplikowanych systemów mikroprocesorowych. Dzięki symulacji istnieje możliwość zlokalizowania i usunięcia błędów popełnionych w trakcie projektowania, zanim zostanie wykonana płyta drukowana.

dokończenie na s. 25

Nowe książki

dokończenie ze s. 9

D. M. Cleal, N. O. Heaton: Knowledge-Based Systems. Implifications for Human-Computer Interfaces. Ellis Horwood, marzec 1988, s. 254

W książce przedstawia się zagadnienia związane z budową i wykorzystaniem systemów z bazą wiedzy, takie jak: tryby interakcji i projektowanie dialogu, możliwość oparcia go na języku naturalnym, inteligentne sprzęgi, wyjaśnianie i funkcje type „*help*", gromadzenie wiedzy, analizę przypadków (ang. *case studies*), rola użytkownika w projektowaniu systemu ekspertowego. Jest także przegląd narzędzi dostępnych na rynku.

G. A. Ringland, D. A. Duce (eds.): Approaches to Knowledge Representation. An Introduction. Research Studies Press, marzec 1988, s. 268

Przegląd metod reprezentacji wiedzy pod kątem umożliwiania odpowiedniego wyboru podczas projektowania systemów z bazą wiedzy. Obaj wydawcy pracują w Rutherford Appleton Laboratory w Wielkiej Brytanii, co może świadczyć o zainteresowaniu tą tematyką w środowisku fizyków.

G. Winstanley: Program Desion For Knowledge Baseд Systems. Sigma Press, luty 1988, s. 228

Autor stara się pokazać jak należy projektować programy w LISPie przeznaczone dla systemów z bazą wiedzy. W projektowaniu uwzględnia się zarówno podejście oparte na sterowaniu celem (ang. *goal driven*), jak i podejście oparte na sterowaniu danymi (ang. *data driven*).

I. Kononenko, N. Lavrac: Prolog Through Examples. A Practical Programming Guide. Sigma Press, czerwiec 1988, s. 200

Kurs programowania w Prologu uwzględniający: podstawowe koncepcje tego języka, style programowania, techniki wyszukiwania błędów, operowanie strukturami danych, rachunek symboliczny, analizę składniową, pisanie interpretera i systemów ekspertowych. Zawiera programy o różnorodnych zastosowaniach.

R. Lucas: Database Applications Using Prolog. Ellis Horwood, czerwiec 1988, s. 160

Książka przeznaczona dla programistów i osób tworzących systemy ekspertowe, w której autor pokazuje, jak można powiązać Prolog z systemami baz danych. Czyni to na trzech przykładach zastosowań: analizie awarii w sieci o dużej skali, identyfikacji włamywaczy w zagadce kryminalnej, wyborze farby w taśmowym systemie malowania samochodów.

J. R. Walters, N. R. Nielsen: Crafting Knowledge Based Systems. Export Systems Made Realistic. J. Wiley, marzec 1988, s. 352

Autorzy (Walters z DEC, Nielsen z SRI International) opisują pełny proces budowania systemu z bazą wiedzy – od samego początku do jego eksploatacji, szczegółowo zajmując się fazami planowania, projektowania, budowania i testowania. Przedstawiają sposoby reprezentacji wiedzy, uwzględniając także i te mniej znane. Wyróżniają reprezentacje: regułową, opartą na ramach, wielokrotnego kontekstu (ang. *multiple context*), modelową (ang. *model-based*) i tablicową.

W. Goshawke, I. D. K. Kelly, J. D. Wigg: Computer Translation of Natural Language. Sigma Press, luty 1988, s. 275

Autorzy zajmują się maszynowym tłumaczeniem języka naturalnego. Po wprowadzeniu w problematykę, przedstawiają uproszczoną metodę liczbowego kodowania tekstu źródłowego przed jego właściwym przetłumaczeniem (metodę tę można wykorzystać w mikrokomputerach). Przykłady dotyczą tłumaczeń prostych zdań z angielskiego na francuski i odwrotnie.

MEDCOM

ZASILANIE AWARYJNE

ZASILACZE BEZPRZERWOWE

naszej produkcji

UPS–200
UPS–300

produkcji tajwańskiej

UPS–500

produkcji japońskiej

GENERATORY PRĄDOTWÓRCZE

różnych mocy

EO/3/90

Biuro
Al. Ujazdowskie 26/39
00–478 Warszawa tel. 28 93 57, teleks 81 70 60

ZAKŁAD PRODUKCYJNO-USŁUGOWY
ul. Wołodyjowskiego 41
02–724 Warszawa tel.43 04 41

JANUSZ ZALEWSKI
Instytut Energii Atomowej
Otwock-Świerk

Okres istnienia oprogramowania (3)

W trzeciej części artykułu omówiono przykłady procesów poprodukcyjnych i integracyjnych oraz możliwości użycia norm i zaleceń w niektórych dziedzinach zastosowań systemów komputerowych.

PROCESY POPRODUKCYJNE

Cztery procesy poprodukcyjne, wyróżnione w normie [24] to: instalowanie oprogramowania, eksploatowanie i serwis, wycofanie oprogramowania z użycia oraz konserwacja. Każdy z tych procesów dzieli się na wiele czynności, które wyliczono poniżej:

- **instalowanie oprogramowania,** na co składa się: planowanie instalacji, dystrybucja oprogramowania, instalacja, zakładanie bazy danych, kontrola oprogramowania w środowisku eksploatacyjnym, prowadzenie aktualizacji;
- **eksploatowanie i serwis,** które obejmują: eksploatację systemu, zapewnienie pomocy technicznej i konsultacji, utrzymanie kroniki wezwań serwisu;
- **wycofanie z użycia,** które dzieli się na: powiadomienie użytkowników, prowadzenie równoległej eksploatacji (dwóch systemów), zakończenie eksploatacji;
- **konserwacja,** polegająca na ponownym zastosowaniu całego cyklu produkcyjnego począwszy od fazy formułowania koncepcji.

Jak widać, procesy poprodukcyjne nie są specjalnie ciekawe dla wykonawców oprogramowania (ang. *developers*), z wyjątkiem procesu konserwacji, który z chwilą podjęcia formalnej decyzji o wprowadzeniu zmian przekształca się w nowy cykl produkcyjny. Nie jest to jednakże typowy punkt widzenia na konserwowanie oprogramowania. Inne podejścia zaprezentowano, na przykład, w [27, 31].

Konserwacja oprogramowania według NBS

W bardzo ciekawym raporcie Krajowego Urzędu Normalizacyjnego USA (*National Bureau of Standards – NBS*) [27] omówiono techniczne i organizacyjne aspekty konserwacji oprogramowania oraz przedstawiono propozycje i zalecenia prowadzące do udoskonalenia tego procesu, a co za tym idzie – do polepszenia jakości oprogramowania. Przez **konserwację oprogramowania** rozumie się *wszystkie te czynności wymagane do utrzymania oprogramowania w eksploatacji, które są wykonywane po odbiorze oprogramowania i wprowadzeniu go do eksploatacji.* Konserwacja oprogramowania wiąże się więc z dokonywaniem zmian w pierwotnie przyjętym oprogramowaniu. Zmiany te mogą polegać na poprawianiu, wstawianiu, usuwaniu, rozszerzaniu i wzbogacaniu oprogramowania.

Z przyczyn technicznych wyróżniono trzy rodzaje konserwacji (podział ten jest stosowany już od kilkunastu lat):

- **konserwacja korekcyjna,** polegająca na wprowadzaniu zmian wynikających z zauważonych defektów oprogramowania, powodujących błędy w działaniu,
- **konserwacja adaptacyjna,** związana ze zmianami w środowisku, w którym oprogramowanie jest eksploatowane,
- **konserwacja perfekcyjna,** dotycząca wszystkich zmian spowodowanych zwiększonymi wymaganiami użytkowników. Dodatkowo wspomniano też o tzw.:
- **konserwacji prewencyjnej,** określając ją jako zapobiegawcze udoskonalanie oprogramowania w celu zoptymalizowania go oraz wyeliminowania zauważonych braków i wad.

Przyczyny wymagające stosowania konserwacji korekcyjnej podzielono na defekty projektowe, logiczne i defekty kodowania. Jako przykładowe źródła zmian wymagających konserwacji adaptacyjnej wyliczono: regulacje prawne, konfigurację sprzętu, formaty danych i struktury plików, oprogramowanie podstawowe (system operacyjny, kompilatory, programy usługowe). Do przyczyn powodujących konieczność prowadzenia konserwacji perfekcyjnej zaliczono głównie zmiany w specyfikacji, np. wzrost wymagań wydajnościowych itp.

W celu głębszego zrozumienia procesu konserwacji wyróżniono kilkanaście kolejnych kroków, upraszczających jego prowadzenie:

- określenie potrzeby zmian,
- dostarczenie żądania zmian (tj. formalnego dokumentu),
- analiza wymagań (zawartych w powyższym dokumencie),
- przyjęcie lub odrzucenie żądania zmian,
- opracowanie harmonogramu prac,
- analiza projektu,
- przegląd projektu,
- zmiana i uruchamianie kodu,
- przegląd proponowanych zmian kodu,
- testowanie,
- uaktualnienie dokumentacji,
- lustracja,
- odbiór przez użytkownika,
- poinstalacyjny przegląd zmian i ich wpływu na cały system,
- zakończenie prac.

Nie ulega wątpliwości, że przebieg tego procesu nie musi być liniowy, tzn. w podanej kolejności; dopuszczalne są iteracje i powtarzanie określonych czynności.

Podane dalej zasady zalecane do stosowania w procesie konserwacji podzielono na cztery grupy:

1) ogólne reguły kontroli procesów konserwacji korekcyjnej, adaptacyjnej i perfekcyjnej;
2) zalecenia techniczne, obejmujące:
- zasady tworzenia kodu źródłowego, jak użycie jednego języka wysokiego poziomu, konwencje kodowania, strukturalizacja i modularyzacja programu, standardowe definiowanie danych, obszerne komentowanie kodu, unikanie rozszerzeń kompilatora,
- zasady dokumentowania programów,
- zalecane techniki kodowania i dokonywania przeglądów, jak kodowanie zstępujące lub wstępujące (ang. *top-down, bottom-up*), przeglądy wzajemne (ang. *peer reviews*), inspekcje, zespół głównego programisty,
- zasady kontrolowania zmian obejmujące żądanie zmian, lustrację kodu (ang. *code audit*, rozumianą jako procedurę służącą do określania, w jakim stopniu kod spełnia normy i praktyki kodowania oraz specyfikację projektową) oraz przegląd i przyjęcie zmian,
- zasady testowania;

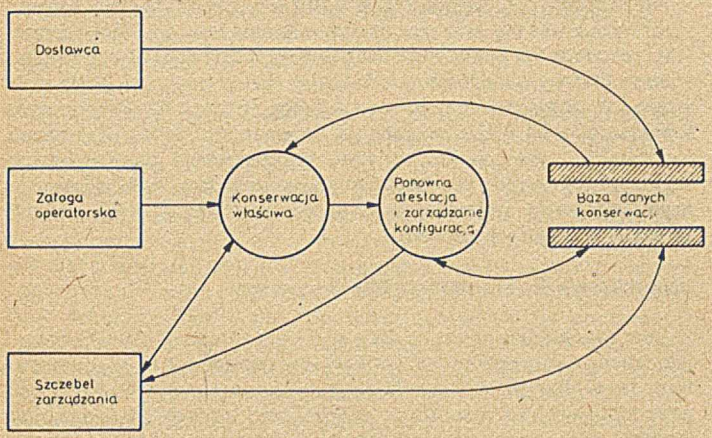
3) narzędzia szczególnie użyteczne w konserwacji oprogramowania, np. analizator odwołań (ang. *cross referencer*), komparator, programy diagnostyczne, biblioteki usługowe i dokumentacyjne, interakcyjne programy diagnostyczne, biblioteki usługowe i dokumentacyjne, interakcyjne programy uruchomieniowe, generatory danych testowych i wiele innych programów;
4) zasady zarządzania, obejmujące określenie celu konserwacji, strategii osiągnięcia tego celu i obsady kluczowych stanowisk prowadzących do jego realizacji.

Konserwacja oprogramowania według EWICS

Jeden z rozdziałów najnowszych zaleceń komitetu ds. niezawodności i bezpieczeństwa EWICS (ang. *European Workshop on Industrial Computer Systems*) dotyczy konserwowania i modyfikowania systemów komputerowych z uwzględnieniem bezpieczeństwa [31, 32]. Muszą one więc stanowić wzmocnienie wszelkich dotychczas znanych norm i praktyk dotyczących konserwowania oprogramowania, gdyż będzie ono przeznaczone do zastosowań szczególnie krytycznych. Tak też jest w rzeczywistości, co widać już po kształcie dokumentu, który przedstawia zalecenia w sposób możliwie sformalizowany. Duży stopień precyzji zaleceń wydaje się szczególnie istotny w konserwacji przede wszystkim z tego względu, że systemy stosowane w zagadnieniach krytycznych na ogół nie mogą być w pełni przetestowane przed wprowadzeniem do eksploatacji.

Jest charakterystyczne, że w wymienionym dukumencie przyjęto także znany już, objęty nawet normami terminologicznymi, podział procesu konserwacji na konserwację prewencyjną, korekcyjną, perfekcyjną i adaptacyjną. Wyodrębniono przy tym cztery elementy składowe procesu konserwacji, od których zależy zapewnienie jego powodzenia.

Głównym wyróżnionym elementem jest uwzględnienie aspektów bezpieczeństwa, dokonywane trójpoziomowo przez szczeble kierownicze użytkownika, wewnętrzną komórkę ds. bezpieczeństwa systemów i podobny organ zewnętrzny. Przykładowo, użytkownicy mogą przeprowadzić konserwację prewencyjną i korekcyjną samodzielnie, ale w przypadku konserwacji perfekcyjnej i adaptacyjnej nie mogą tego zrobić bez odwołania się do komórki wewnętrznej lub zewnętrznego organu ds. bezpieczeństwa systemów. Drugim elementem mającym wpływ na powodzenie konserwacji jest dobór technik zarządzania, np. dotyczących ustalania celów i priorytetów, stosowania norm i zaleceń, dokumentowania procesu konserwacji, przeprowadzania lustracji i przeglądów. Pozostałe elementy to dobór narzędzi oraz doświadczenie personelu technicznego.

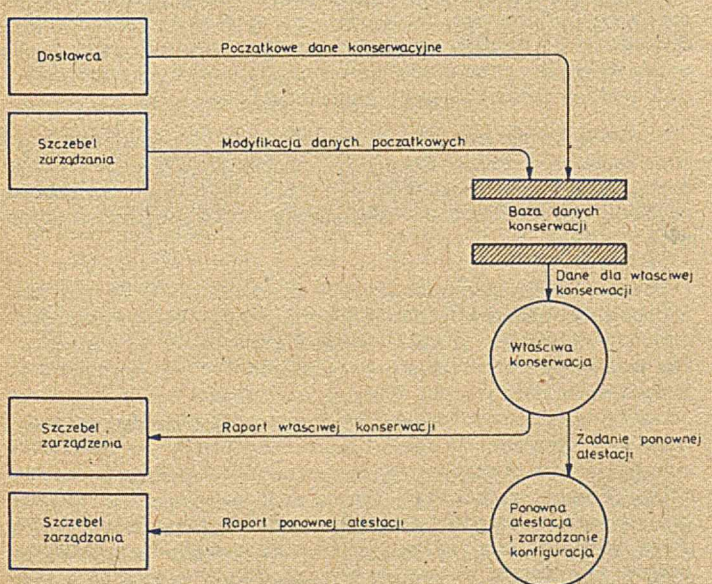


Rys. 1 Zasada procesu konserwacji

Techniczne podejście do procesu konserwacji oparto na formalizmie opisującym cały proces jako cyklicznie wykonywany ciąg czynności, rozpoczynający się od żądania zmian a kończący wydaniem sprawozdania z konserwacji, co przedstawia się w postaci diagramu przepływu danych (ang. *data flow diagram*). Na najwyższym poziomie abstrakcji (rys. 1) wyróżniono tylko dwie główne czynności, wykonywane w każdym cyklu konserwacyjnym:

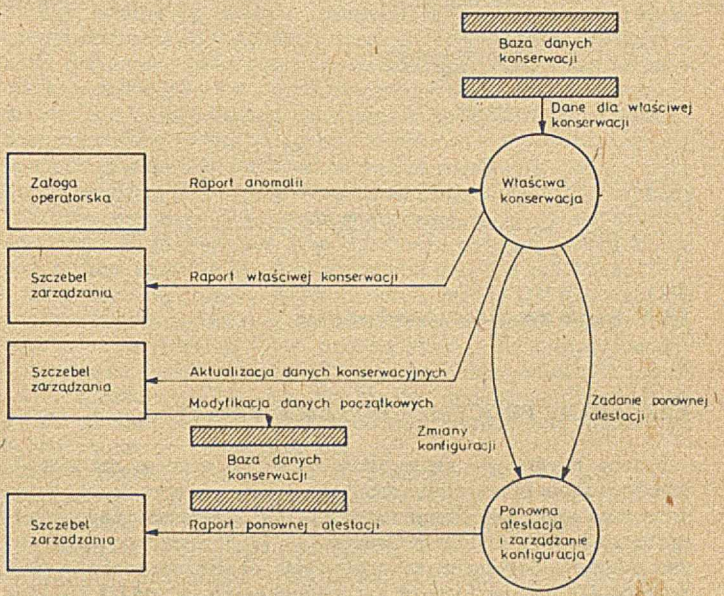
– **właściwa konserwacja,**

– **ponowna atestacja i zarządzanie konfiguracją,** polegające na ponownym przetestowaniu systemu i zebraniu danych o wykonanych czynnościach oraz zmianach dokonanych w konfiguracji systemu. Trzecim, oprócz tych czynności, niezwykle ważnym ogniwem procesu konserwacji jest **baza danych,** zawierająca wszystkie dane niezbędne do wykonania konserwacji, szczegóły konfiguracji (pierwotnej i zmienionej) oraz zapisy historii systemu i procesu.



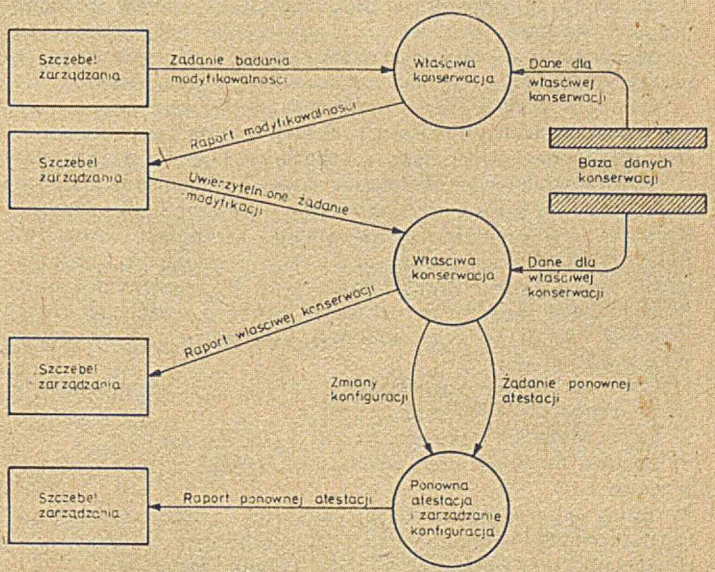
Rys. 2 Konserwacja prewencyjna

Tak rozumiany proces konserwacji opisano w zaleceniach oddzielnie dla każdego rodzaju konserwacji: prewencyjnej, korekcyjnej oraz – łącznie – perfekcyjnej i adaptacyjnej, rozróżniając dodatkowo trzy poziomy szczegółowości. Opis na pierwszym poziomie, przedstawiony na rys. 2-4, zawiera tylko część istotnych informacji, mających ogólny charakter. Przykładowo, inicjowanie konserwacji prewencyjnej następuje na podstawie dyspozycji pochodzącej z bazy danych, zgodnie z uprzednio ustalonym harmonogramem. Zainicjowanie konserwacji korekcyjnej może nastąpić jako konsekwencja przedstawionego przez operatora raportu o zauważonych anomaliach w działaniu obiektu.



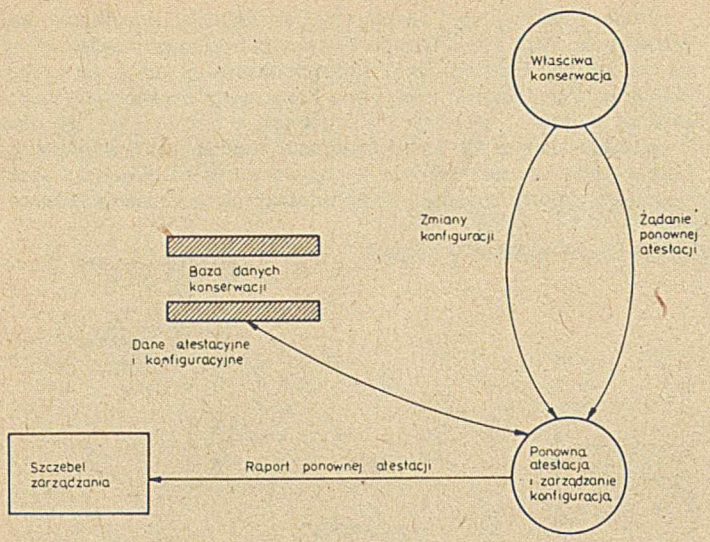
Rys. 3 Konserwacja korekcyjna

Konserwacja perfekcyjna i adaptacyjna może być zainicjowana przez szczeble kierownicze w postaci żądania badań możliwości modyfikacji. Każdy cykl konserwacji kończy się czynnościami atestacji i zarządzania konfiguracją (rys. 5), mającymi na celu stwierdzenie, że system po konserwacji spełnia wymagania funkcjonalne. Istotą tych czynności jest przetestowanie systemu i uaktualnienie dokumentacji; ich opis podano również na kilku poziomach szczegółowości.



Rys. 4 Konserwacja perfekcyjna i adaptacyjna

W zaleceniach podkreśla się, że każda proponowana zmiana w eksploatowanym systemie musi być najpierw poddana dokładnej analizie, a następnie zostać zaakceptowana na odpowiednim szczeblu zarządzania. Jeżeli zmiana ma być zrealizowana, to należy ją traktować jako niezależne przedsięwzięcie i stosować sprawdzone metody projektowania i wytwarzania. Na zakończenie należy dokonać ponownej atestacji systemu i zaktualizować jego dokumentację. Podany w zaleceniach precyzyjny słownik danych oraz minispecyfikacje poszczególnych czynności konserwacyjnych czynią te zalecenia bardzo użytecznymi w praktyce.



Rys. 5 Ponowna atestacja i zarządzanie konfiguracją

PRZYKŁADY PROCESÓW INTEGRACYJNYCH

Procesy integracyjne nie są niezbędne do skutecznego zakończenia przedsięwzięcia programistycznego. Dwa z tych procesów odgrywają szczególną rolę – **weryfikacja i atestacja** oraz **zarządzanie konfiguracją** oprogramowania, dlatego poświęcono im (w każdym przypadku) po dwa oddzielne dokumenty normalizacyjne, określające zarówno same zalecenia [15, 18], jak i sposób korzystania z tych zaleceń [21,22]. Inny równie ważny proces, jakim jest **wytwarzanie dokumentacji,** ma nieco inne oblicza w różnych fazach okresu istnienia oprogramowania, dlatego poświęcono mu kilka oddzielnych norm.

Proces weryfikacji i atestacji według IEEE

Proces weryfikacji i atestacji obejmuje zaplanowanie i wykonanie wszystkich przeglądów, lustracji, testów i ocen, przeprowadzanych w okresie istnienia oprogramowania. Przez **weryfikację** (ang. *verification*) rozumie się tu *proces stwierdzania, czy wynik określonej fazy cyklu produkcyjnego oprogramowania spełnia wymagania ustalone w poprzedniej fazie.* **Atestacja** (ang. *validation*) oznacza natomiast *proces oceny oprogramowania na końcu cyklu produkcyjnego, w celu stwierdzenia, czy produkt jest zgodny z wymaganiami zawartymi w specyfikacji.*

Norma [18], przyjmując punkt widzenia przedstawiony we wcześniejszych dokumentach normalizacyjnych [28], reprezentuje podejście sekwencyjne do procesu weryfikacji i atestacji, zalecając dla cyklu produkcyjnego oprogramowania stosowanie modelu kaskadowego, opisanego w [14], z podziałem na fazy: *formułowania koncepcji, specyfikowania wymagań, projektowania, implementowania, testowania, instalowania i kontroli, eksploatacji i konserwacji,* połączone nadrzędną fazą *zarządzania,* trwającą przez cały cykl produkcyjny. Podstawowy dokument, zwany **planem weryfikacji i atestacji,** powinien według tej normy opisywać dla każdej fazy oddzielnie następujące aspekty procesu weryfikacji i atestacji:

1) określenie celu planu, zawierające opis przedsięwzięcia programistycznego i produktów finalnych;
2) nazwy wiążących dokumentów (norm, kontraktów itp.);
3) definicje terminów oraz wyjaśnienia skrótów i notacji użytych w planie;
4) ogólny opis procesu weryfikacji i atestacji, z uwzględnieniem zasad organizacyjnych, harmonogramu prac, niezbędnych zasobów, przydziału zadań oraz zalecanych narzędzi, technik i metodyki;
5) właściwy plan charakteryzujący przebieg procesu zgodnie z podziałem na fazy według modelu kaskadowego (najważniejsza część dokumentu);
6) zasady prowadzenia sprawozdań z przebiegu weryfikacji i atestacji w całym cyklu produkcyjnym;
7) procedury administracyjne niezbędne do zrealizowania planu.

Główna część planu (pkt 5) dotyczy szczegółowego przebiegu procesu i określa następujące istotne składniki, które powinny być opisane dla każdej fazy oddzielnie;

1) zadania (weryfikacji i atestacji w tej fazie),
2) metody i kryteria (użyte do realizacji powyższych zadań),
3) dane wejściowe i produkty wyjściowe,
4) harmonogram (realizacji zadań),
5) zasoby (niezbędne do realizacji zadań),
6) założenia i ryzyko (występujące przy realizacji),
7) podział ról i odpowiedzialności (między wykonawcami zadań).

Dla każdej fazy podano też wykaz wymaganych dokumentów źródłowych i wynikowych (określonych po angielsku jako *inputs* i *outputs*) oraz minimalny zestaw zadań (czynności) weryfikacyjno-atestacyjnych koniecznych do zrealizowania przy wytwarzaniu oprogramowania dla systemów krytycznych. Podstawowe zadania dotyczą analizy *śladowalności* (tj. umiejscowionych w poprzednich fazach przyczyn i źródeł przyjętych rozwiązań), oceny produktów cząstkowych (powstających w każdej fazie), analizy sprzężeń, planowania testów i wykonania różnego rodzaju testów, lustracji i przeglądów.

Nowa norma [22], będąca jeszcze w stadium opracowywania, zawiera zbiór wskazówek dla osób stosujących zalecenia zawarte w dokumencie [18] i rozszerza ten punkt widzenia o podejście czynnościowe, przedstawione w [24], wyrażające cykl produkcyjny oprogramowania przez wiele wzajemnie powiązanych procesów, składających się z bardziej szczegółowych czynności. Najważniejszym skutkiem tego podejścia dla procesu weryfikacji i atestacji jest wyraźne wyodrębnienie poszczególnych czynności testowania, tzn. testowania komponentów, testu integracyjnego, testowania systemu i testu odbiorczego, dla których podano szereg szczegółowych zaleceń.

Inne zalecenia dla procesu weryfikacji i atestacji

Dwa wcześniejsze zalecenia dla weryfikacji i atestacji oprogramowania, norma amerykańska [28] i dokumenty europejskie [8, 9] opublikowane też w zbiorze [30], nie odbiegają zbytnio w treści od norm IEEE, gdyż są oparte na wspólnych założeniach, których podstawą jest istnienie cyklu produkcyjnego oprogramowania z wyraźnie wyodrębnionymi fazami.

Istotną cechą pierwszej normy jest podział wszystkich czynności wykonywanych w okresie istnienia oprogramowania na czynności produkcyjne i czynności weryfikacyjno-atestacyjne oraz wydzielenie i zidentyfikowanie ich wyników. Nie dającą się przecenić konsekwencją tego spostrzeżenia jest wprowadzony już do praktyki programistycznej *całkowity rozdział zespołu produkcyjnego od zespołu weryfikacyjno-atestacyjnego,* który najczęściej jest kontrahentem zewnętrznym. Do podstawowych czynności weryfikacyjno-atestacyjnych zaliczono:

- **w fazie specyfikowania wymagań** – opracowanie planu weryfikacji i atestacji, początkowa specyfikacja testu, przegląd i analiza wymagań, przegląd i analiza podręczników użytkowania,
- **w fazie projektowania** – przegląd i analiza projektu, zaimplementowanie lub nabycie narzędzi wspomagających testowanie,
- **w fazie kodowania i testowania** – końcowa specyfikacja testu, przegląd i analiza kodu, przetestowanie kodu,
- **w fazie instalowania** – przeprowadzenie testu odbiorczego,
- **w fazie eksploatacji i konserwacji** – ocena oprogramowania, testowanie regresywne (ang. *regression testing,* polegające na wykonaniu testów sprawdzonych i działających poprawnie dla poprzednich wersji kodu, w celu wykrycia – przez porównanie wyników – defektów wprowadzonych podczas modyfikowania), ocena modyfikacji kodu.

Ważne są również zawarte w tej normie sugestie i przykłady dotyczące stosowania odpowiednich technik. Z wymienionych względów normę tę można uznać za pionierską w propagowaniu właściwego spojrzenia na problematykę weryfikowania i atestowania produktów programistycznych.

To samo podejście zaprezentowano w zaleceniach [8, 9]. Dotyczą one, odpowiednio, metodyki i technik weryfikowania i atestowania systemów komputerowych w zastosowaniach krytycznych, przede wszystkim pod względem rzetelności oprogramowania (ang. *dependability*). Wskazano tu ponownie, że *główną i niezbywalną zasadą weryfikacji i atestacji jest całkowita niezależność i rozłączność zespołów: wykonawczego oraz weryfikacyjno-atestacyjnego.* **Weryfikacja** – w rozumieniu tego dokumentu – *polega także na porównaniu zgodności wyniku każdego kolejnego etapu wytwarzania systemu komputerowego z ustaleniami dokonanymi na poprzednim etapie.* **Atestacja** natomiast *stanowi stwierdzenie zgodności całego systemu przekazywanego do eksploatacji z wymaganiami określonymi w jego specyfikacji.* Zalecenia podane w tych rozdziałach stanowią szereg wskazówek co do planowania i stosowania preferowanych technik w kolejnych fazach wytwarzania wyrobu, jak: *specyfikowanie, projektowanie, kodowanie, testowanie modułów, integrowanie modułów, integrowanie sprzętu i oprogramowania i atestowanie całego systemu.*

Testowanie modułów (jednostek, komponentów)

Spośród wielu rodzajów testowania, jednej z głównych czynności w procesie weryfikacji i atestacji, wyróżniono testowanie modułów (zwanych inaczej komponentami lub jednostkami programowymi) i poświęcono mu oddzielną normę [17]. Wydzielono w niej dwie fazy testowania, tj. przygotowanie testu i przeprowadzenie testu, oraz osiem czynności składających się na te fazy. **Przygotowanie testu** polega na:
- zaplanowaniu podejścia, wykorzystania zasobów i harmonogramu,
- określeniu testowanych właściwości (na podstawie specyfikacji),
- uszczegółowieniu planu,
- zaprojektowaniu zestawu testów,
- zaimplementowaniu szczegółowego planu i projektu.

Na **przeprowadzenie testowania** składają się następujące czynności:
- wykonanie testów,
- kontrola zakończenia testów,
- ocena wyników.

Obie fazy i ich czynności przedstawiono w normie w postaci diagramów przepływu danych, wyliczając dokumenty źródłowe i wynikowe, co znacznie ułatwia jej stosowanie.

Testowanie rozumie się tu (u w a g a !) jako *celowe wprowadzenie defektów do programu w celu spowodowania awarii;* mniej ważne jest stwierdzenie poprawnej pracy programu (!), chodzi bowiem przede wszystkim o sprawdzenie działania programu w sytuacjach nieoczekiwanych. Tak rozumiane testowanie jest przeciwieństwem **uruchamiania** (ang. *debugging*), które jest czynnością polegającą na *analizowaniu awarii w celu zlokalizowania i poprawienia defektów w programie.* Dwoma podstawowymi dokumentami powstającymi w czasie testowania są: **specyfikacja projektu testu** i **sprawozdanie z testowania.** W normie podkreślono wyraźnie, że nie jest jej celem zalecanie określonych technik lub narzędzi testowania, choć podano najważniejsze publikacje, w których można znaleźć odpowiednie opisy.

Przeglądy i lustracje

Do innych ważnych czynności wykonywanych w procesie weryfikacji i atestacji oprogramowania należą przeglądy i lustracje, dlatego poświęcono im niezależną normę [19]. **Przeglądem** (ang. *review*) nazywa się tu *postępowanie, podczas którego produkt programowy przedstawia się zainteresowanym* (np. członkom zespołu, kierownictwu, użytkownikom, nabywcom itp.), *w celu zebrania opinii lub odbioru,* a **lustracją** (ang. *audit*) – *niezależne sprawdzenie produktu programowego w celu oceny jego zgodności ze specyfikacją, normami, kontraktem lub innymi kryteriami* [14].

W normie zidentyfikowano trzy rodzaje przeglądów, **przegląd zewnętrzny** (ang. *management review*), **przegląd techniczny** (ang. *technical review*), **inspekcję oprogramowania** (ang. *software inspection*) oraz cztery rodzaje lustracji – **lustracja systemu jakości** (ang. *quality systems audit*), **lustracja funkcjonalna** (ang. *functional configuration audit*), **lustracja fizyczna** (ang. *functional configuration audit*) i **lustracja produkcyjna** (ang. *inprocess audit*). Wszystkie rodzaje przeglądów opisano według następującego schematu:
- cel, streszczenie i założenia,
- kryteria zewnętrzne, zasoby i odpowiedzialność,
- dokumenty wejściowe, procedury i dokumenty wyjściowe,
- kryteria rozpoczęcia i zakończenia,
- podatność na lustracje (ang. *auditability*).

Lustracje opisano podobnie, koncentrując się na elementach wspólnych – głównie procedurach. Przeglądy znane z innych procesów, np. stosowane w zapewnianiu jakości (por. cz. 1 artykułu), tj. *przegląd wymagań, wstępny i krytyczny przegląd projektu,* czy *przegląd planu weryfikacji i atestacji,* potraktowano tu jak specyficzne zadania, które można wykonać posługując się zasadami opracowanymi w tym dokumencie jako zaleceniami rodzajowymi.

Proces zarządzania konfiguracją

Celem **zarządzania konfiguracją** oprogramowania w cyklu produkcyjnym jest *utrzymywanie informacji o aktualnym składzie wytwarzanego oprogramowania i kontrolowanie jego rozwoju.* Dzięki temu uzyskuje się możliwość odtworzenia źródeł pewnych decyzji, przyczyn konkretnych rozwiązań, śledzenia istotnych właściwości oprogramowania, a więc – cechę, którą można nazwać śladowalnością. Podstawowym dokumentem dla tego procesu jest **plan zarządzania konfiguracją** (ang. *software configuration management plan, SCMP*), opisany w normie [15]. Wyróżniono w nim cztery zasadnicze czynności wykonywane przy prawidłowym prowadzeniu tego procesu:
- identyfikowanie konfiguracji,
- kontrola konfiguracji,
- rejestrowanie stanu konfiguracji,
- wykonywanie przeglądów i lustracji.

W obecnie przygotowywanej aktualizacji tej normy wyróżniono jeszcze dwie czynności:
- kontrola sprzężeń,
- kontrola podwykonawców i sprzedawców (tzn. poddostawców).

Te dwie ostatnie czynności są szczególnie istotne przy zlecaniu części oprogramowania do wykonania na zewnątrz. Norma zawiera wskazówki nt. przygotowywania planu zarządzania konfiguracją, tzn. określenia, kto, co, kiedy i jak ma robić, aby zapewnić należyte utrzymywanie informacji o konfiguracji oprogramowania. Spośród wielu wprowadzonych pojęć warto zwrócić uwagę na dwa pokrewne. *Wyróżniony element konfiguracji podlegający kontroli nazywa się* **wersją** (ang. *version*). Przy każdej modyfikacji nadaje się wersji nowy identyfikator. Przez **wydanie** (ang. *release*) *rozumie się natomiast proces przekazania elementu konfiguracji będącego półproduktem* (ang. *baseline*) – innej organizacji lub na inny poziom tej samej organizacji – w celu przyjęcia modyfikacji.

Główną zaletą uzupełniających tę normę zaleceń [21] (por. też [33]) jest szczegółowe wyjaśnienie poszczególnych czynności. Każdą z nich rozbito na fazy opisując je za pomocą pytań wskazujących aspekty konieczne do rozważenia przy wykonywaniu tych czynności. Do normy [21] włączono również wiele przykładów dodatkowo ilustrujących zakres jej stosowania. Przykłady dotyczą wytwarzania dokumentu SCMP dla czterech różnych zastosowań:
- krytycznego oprogramowania dla systemów wbudowanych,
- małego systemu budowanego eksperymentalnie,
- organizacji zajmującej się konserwacją oprogramowania,
- systemu sterującego linią produkcyjną.

Użyteczność norm dotyczących zarządzania konfiguracją potwierdzono opracowując zalecenia do stosowania ich w systemach oprogramowania dla elektrowni jądrowych [20].

Proces wytwarzania dokumentacji według norm amerykańskich

Dobre dokumentowanie wszystkich czynności podczas wytwarzania i konserwacji oprogramowania stanowi nieocenione źródło informacji zarówno dla samych wykonawców, jak i dla późniejszych użytkowników tego oprogramowania. Należy pamiętać, że ścisłe rozumienie oprogramowania obejmuje zawsze programy komputerowe wraz z dokumentacją. Ze względu na dużą różnorodność procesów wytwórczych trudno wszakże opracować ogólną normę odnoszącą się do wszystkich czynności. Jedno ze starszych zaleceń, jakie znam [29], traktuje proces dokumentowania jako całość, formułując wiele istotnych wskazówek dotyczących zarządzania tym procesem, pomijając przy tym zagadnienia wykorzystywanych technik i narzędzi.

Ponieważ inny rodzaj dokumentacji jest przydatny w samym wytwarzaniu oprogramowania, a inny w jego eksploatacji, dzieli się ją na **dokumentację cyklu produkcyjnego** i **dokumentację wyrobu,** co w przybliżeniu pokrywa się z podziałem na **dokumentację produkcyjną** i **dokumentację użytkową. Dokumentacja cyklu produkcyjnego** obejmuje *dokumenty będące wynikami poszczególnych faz wytwarzania oprogramowania,* a **dokumentacja wyrobu** (dokumentacja użytkowa) – *wszystkie dokumenty towarzyszące oprogramowaniu w dostawie* (oba te zbiory dokumentów nie muszą być i na ogół nigdy nie są całkowicie rozłączne). Większość przykładowych dokumentów produkcyjnych wyliczono w poprzednich rozdziałach tego artykułu, natomiast do dokumentacji wyrobu zalicza się takie dokumenty, jak *podręcznik użytkowania, podręcznik operatora, podręcznik konserwacji* itd.

Biorąc pod uwagę jakość dokumentacji, zalecono jej klasyfikowanie na czterech poziomach: minimalnym, wewnętrznym, roboczym i publikacyjnym. W zasadzie, jedynie **poziom publikacyjny** dopuszcza rozpowszechnianie dokumentacji poza obrębem macierzystej instytucji, gdyż odnosi się do programów przeznaczonych do powszechnego użycia oraz programów stanowiących przedmiot publikacji naukowych; strona formalna tej dokumentacji musi być zgodna z wymaganiami i normami opracowującej instytucji. **Poziom roboczy** stosuje się do programów używanych przez różne osoby w tej samej instytucji i niekiedy w innych instytucjach; dokumentacja jest sporządzana i rozpowszechniana w postaci maszynopisu z minimalnym opracowaniem edytorskim. **Poziom wewnętrzny** odnosi się do specjalizowanych programów, które nie mają szerszego zastosowania. Ich dokumentacja może ograniczać się do obszernych komentarzy w wydruku programu, aby ułatwiać jego użytkowanie i modyfikowanie; dokumentacja opisowa jest bardzo

skąpa. **Poziom minimalny** jest odpowiedni dla pojedynczych programów powstających na przykład w ciągu miesiąca; dokumentacja ogranicza się do wydruku programu, notatek projektowych, danych testowych i abstraktu programu.

W omawianych zaleceniach zwrócono też uwagę na przyczyny powstawania dokumentacji o złej jakości (niski priorytet dokumentowania, brak wspomagania i niedostatek zasobów, brak planowania, negatywne postawy personelu itp.) i zaproponowano sposoby rozwiązywania odpowiednich problemów przez właściwe zarządzanie procesem wytwarzania dokumentacji, zwracając uwagę m.in. na istnienie norm i dobrych praktyk.

Jedną z norm IEEE dotyczących dokumentowania cyklu produkcyjnego jest norma [16] obejmująca zagadnienia dokumentacji testowania programów. Opisano w niej osiem dokumentów, które powinny powstawać jako wynik czynności testowania. Są to:

- **plan testowania,** rozumiany jako opis zakresu, podejścia, zasobów i harmonogramu zamierzonych działań; określający przedmiot i podmiot testu, testowanie właściwości i zadania testu oraz (co bardzo ważne) ryzyko towarzyszące testom;
- **specyfikacja projektu testu,** obejmująca opis podejścia zastosowanego do testowania określonej właściwości i identyfikująca określone testy;
- **specyfikacja testu,** określająca dane wejściowe, przewidywane wyniki i warunki wykonania testu;
- **specyfikacja procedury testowania**, tj. opis ciągu działań prowadzących do wykonania testu;
- **raport przekazania testu** (ang. *test item transmittal report*), zawierający informację identyfikującą oraz miejsce przechowywania testu i jego status;
- **kronika testu,** obejmująca chronologiczny wykaz istotnych informacji o wykonaniach testu;
- **raport nieprawidłowości** (ang. *test incident report*), opisujący każde wydarzenie podczas testowania wymagające dalszych badań;
- **sprawozdanie z testowania**, podsumowujące przebieg testowania i uzyskane wyniki wraz z oceną poszczególnych testów.

Dokumentacja produktu, a więc dokumentacja użytkowa, ma zasadniczo inne przeznaczenie niż dokumentacja produkcyjna, dlatego ujęto ją w zupełnie innej normie [23]. Norma ta zawiera zalecenia dotyczące przygotowywania dokumentów mających służyć użytkownikom do instalowania i eksploatowania programów. Za najważniejsze uznano precyzyjne określenie na początku: właściwości samego oprogramowania (tj. jego funkcji, sposobów użycia, a nade wszystko sprzężeń użytkowych), kręgu odbiorców dokumentacji, zestawu wymaganych dokumentów i trybów ich użycia. Podział dokumentacji ze względu na tryb użycia wynika z różnic, jakie dzielą **dokumenty podstawowe (instruktażowe),** z których użytkownik uczy się eksploatacji i obsługi programu, od tzw. **dokumentów referencyjnych,** z których korzysta się w celu przypomnienia nazw poleceń, wyjaśnienia komunikatów, zgłębienia szczegółów technicznych itp. Dla obu rodzajów dokumentów podano zalecane zawartości, nieco różniące się między sobą. Jest charakterystyczne, że ta norma nie opisuje treści żadnego konkretnego dokumentu (por. [34]), lecz dostarcza wzorca do tworzenia dowolnej dokumentacji użytkowej, której rodzaj zależy od stosowanej praktyki, od ustaleń między wytwórcą a nabywcą oprogramowania itd.

Dokumentowanie systemu według EWICS

W nieco innej formie sformułowano zalecenia dokumentacyjne w książce [30], której pierwszy rozdział zawiera wymagania dotyczące dokumentowania systemów komputerowych nie tylko podczas ich wytwarzania, lecz także w pełnym okresie istnienia, tzn. w fazach eksploatacji i konserwacji. Choć nie obejmuje on tak ważnych przy wytwarzaniu faz, jak studium realizowalności i projektowanie, pomijając też inny istotny aspekt okresu istnienia systemu – zarządzanie konfiguracją, zawiera wiele ciekawych wskazówek dotyczących dokumentowania systemów komputerowych w zastosowaniach krytycznych, przy założeniu, że taki system stanowi jedynie część większej instalacji. Dlatego też najważniejsze fragmenty tego rozdziału dotyczą zarządzania dokumentacją, opisu realizacji systemu z uwzględnieniem aspektów użytkowych, wielostronnej oceny systemu i zarządzania całym przedsięwzięciem. Poszczególne tematy, odpowiadające wybranym aspektom okresu istnienia systemu, dzielą się na podtematy (np. zarządzanie dokumentacją dotyczy jej struktury, standardów dokumentacyjnych i jej konserwowania), które obejmują najważniejsze elementy składowe, wyliczone w postaci punktów (dzielących się często na podpunkty) jako elementów tzw. list kontrolnych (ang. *checklist*). Przykładowo, jeden z punktów opisujących konserwowanie dokumentacji zaleca stosowanie do tego celu następujących narzędzi: edytorów tekstów, generatorów raportów, generatorów spisu treści i indeksów, komparatorów tekstów, systemów archiwowania, baz danych i poczty elektronicznej.

Może się wydawać, że tak liczne sugestie co do uwzględniania wielu szczegółów dokumentowania są raczej barierą niż ułatwieniem w konkretnych sytuacjach. W rzeczywistości jednak, stanowią one jedynie dobrze uporządkowany układ odniesienia dla wykonawców systemu, którzy sami zadecydują, jakie z nich należy wziąć pod uwagę. Na tę dowolność właśnie kładzie się szczególny nacisk w zaleceniach, ponieważ poszczególne systemy komputerowe o najrozmaitszym przeznaczeniu są tak odmienne, że schematyczne dokumentowanie identycznych aspektów i w jednakowy sposób byłoby pozbawione sensu. Zalecenia, aczkolwiek bardzo szczegółowe, zostawiają jednak duży margines dowolności.

Aby w dokumentowaniu systemu komputerowego uwzględnić jego niezawodność i bezpieczeństwo – cechy trudne do wyodrębnienia i opisania na papierze – autorzy utrzymali ogólny schemat zaleceń, z podziałem na tematy, podtematy i punkty, proponując w poszczególnych miejscach dodatkowe ujęcie tych elementów opisu, które są związane z niezawodnością i bezpieczeństwem systemu. Na przykład, w opisie systemu jako całości występują oddzielne listy kontrolne, złożone z punktów odnoszących się kolejno do zintegrowanego systemu (opis i ograniczenia zapewnienia bezpieczeństwa, obsługa defektów, zasady konserwacji i prowadzenia napraw, ostrzeganie przed niesprawnościami), samego sprzętu (wielopunktowy program zapewnienia niezawodności i odporności na defekty) i samego oprogramowania (osiąganie bezpieczeństwa i zabezpieczanie przed nieuprawnionym dostępem). Niezwykle ważną cechą zaleceń jest dostarczenie niemal pełnej informacji o możliwościach wytwarzania dokumentacji i korzystania z niej. Po zapoznaniu się z tą partią materiału, wykonawca lub użytkownik powinien nabyć ogólne umiejętności tworzenia dokumentacji w całym okresie istnienia systemu, przy utrzymaniu dużej dowolności w generowaniu rzeczywiście potrzebnych dokumentów, uwzględniających elementy niezawodności i bezpieczeństwa.

ZASTOSOWANIE NORM

Zastosowanie omówionych norm i zaleceń można rozumieć dwojako. Po pierwsze, mogą one stanowić *materiał źródłowy do bardziej szczegółowych opracowań normalizacyjnych* w wielu dziedzinach wykorzystania systemów komputerowych. Szczególnie istotna jest ich rola przy wytwarzaniu tzw. oprogramowania krytycznego, np. w systemach wojskowych [4], kosmicznych [7] jądrowych [1,11,13,20] i przemysłowych [30,31]. Bardziej bezpośrednie korzyści przynosi *użycie tych norm w praktycznym kwalifikowaniu programów komputerowych pod względem jakości.*

W najbliższej mi dziedzinie wytwarzania oprogramowania do zastosowań jądrowych od kilku lat stosuje się kwalifikowanie programów *na poziomie wewnętrznym* (producenta, kontraktu itp.) lub *na poziomie ogólnokrajowym* (licencjonowanie). W jednej z najważniejszych prób badawczych atestacji oprogramowania przeznaczonego dla elektrowni jądrowych [6], cykl produkcyjny prowadzono według reguł inżynierii oprogramowania oddzielnie przez dwa zespoły wykonawców (uzyskując dzięki temu dodatkowo *efekt zróżnicowania*, ang. *diversity*), posługując się specyfikacją formalną, programowaniem strukturalnym, statycznymi narzędziami testowania i automatycznym komparatorem w *łożu testowym* (ang. *testbed*). Cały cykl był nadzorowany przez niezależny zespół weryfikacyjno-atestacyjny, który wykonywał przeglądy specyfikacji, rewizje kodu i testy weryfikacyjne. Dzięki przyjęciu takiej metody pracy uzyskano zdaniem autorów przedsięwzięcia następujące pozytywne skutki:
– znaczne zwiększenie prawdopodobieństwa wykrycia defektów z chwilą ich wprowadzenia,
– zmniejszenie kłopotów związanych z doborem sprzężeń programowych,
– ogólne zwiększenie poprawności oprogramowania,
– zmniejszenie względnego kosztu jednego wiersza kodu (mimo komplikacji związanych z większą liczbą wykonawców).

Licencjonowanie systemów komputerowych stosowanych w elektrowniach jądrowych jest wymagane przez ustawodawstwo wielu krajów.

Nie jest jednak jeszcze rozwiązana jednoznacznie sprawa metodyki badań prowadzących do wydania licencji, choć wszystkie podejścia są oparte w mniejszym lub większym stopniu na przedstawionym modelu cyklu produkcyjnego. Przykładowo, włoski urząd ENEA opiera się na definicji jakości oprogramowania IEEE [5], co prowadzi do konieczności wyodrębnienia i oceny poszczególnych cech jakości (atrybutów oprogramowania). Uznano, że możliwe jest wydzielenie takich cech dla każdej z określonych klas systemów komputerowych według podziału zalecanego w normie IEC [12]. Ponieważ zastosowania dotyczą systemów bezpieczeństwa, określenie cech następuje na drodze **wstępnej analizy bezpieczeństwa** (ang. *preliminary safety analysis*), wykonywanej przez wnioskodawcę na podstawie zaleceń wydanych przez ten urząd (ogólne kryteria projektowe). Ze względu na brak dostatecznie dobrych i powszechnie uznanych metryk służących do oceny tych cech, stosuje się procedury weryfikacyjno-atestacyjne oparte na analizie cyklu produkcyjnego, głównie w fazie projektowej.

Znany urząd zachodnioniemiecki TÜV (niem. *Technischer Überwachungs-Verein*) opracował własną procedurę licencjonowania systemów komputerowych czasu rzeczywistego z uwzględnieniem bezpieczeństwa [10], która objęła oprócz przemysłu jądrowego także naftowy, chemiczny i in. Punktem wyjścia dla tych prac był również podział zastosowań na klasy pod względem ich wagi dla bezpieczeństwa. Dla każdej z klas sformułowano następnie inny zestaw kryteriów jakości (w postaci cech, które powinien posiadać licencjonowany system). Poszczególnym kryteriom (cechom) przyporządkowano odpowiednie listy kontrolne w postaci pytań, służące do sprawdzania jakości podczas całego cyklu produkcyjnego. Kontroli podlegają: specyfikacja wymagań, projekt systemu i zrealizowany system. Ta metodyka jest w dalszym ciągu rozwijana, głównie pod względem doboru automatycznych narzędzi wspomagających, których brak szczególnie się odczuwa. Na podstawie raportu TÜV wnioskodawca może otrzymać licencję, co z kolei (interesujący szczegół) może zostać zaskarżone do sądu. Podobną metodykę stosuje się w innych krajach, np. we Francji [3], w innych dziedzinach zastosowań, np. w transporcie [25], ze względu na szczególne wymagania dotyczące bezpieczeństwa, dążąc nawet do koordynacji na szczeblu międzynarodowym [2, 26].

LITERATURA

[1] ANS: Application Criteria for Programmable Digital Computer Systems in Safety Systems of Nuclear Power Generating Stations. ANS Std 7-4.3.2-1982, American Nuclear Society, La Grange Park (IL), July 1982

[2] Bloomfield R. E., Ehrenberger W. D.: Licensing issues associated with the use of computers in the nuclear industry. Report EUR-11147, Commission of the European Communities, Luxembourg, 1987

[3] Colling B. et al.: Digital Integrated Protection System SPIN Qualification Process. Report IAEA SM-265/58. International Atomic Energy Agency, Vienna, 1984

[4] DOD: Military Software Development. DOD-Std-2167. US Department of Defense, Washington (DC), 1984

[5] Di Marco G., Pasquini A.: Licensing of Real Time Systems in Nuclear Power Plants. Proc. Conf. Hardware and Software for Real Time Process Control. J. Zalewski, W. Ehrenberger (Eds.), pp. 567–568, North-Holland, Amsterdam, 1989

[6] EPRI: Validation of Real-Time Software for Nuclear Plant Safety Applications. Report NP-2646, Electric Power Research Institute, Palo Alto (CA), November 1982

[7] ESA: Software Engineering Standards. European Space Agency, Darmstadt, 1983

[8] EWICS TC7: Guidelines for Verification and Validation of Safety-Related Software. Computers and Standards, Vol. 4, pp. 33–41, 1985

[9] EWICS TC7: Techniques for Verification and Validation of Safety-Related Software. Computers and Standards, Vol. 4, pp. 101–112, 1985

[10] Glöe G., Westhäusser R.: Assessment and Licensing of Control Computers in the FRG. Proc. Conf. Hardware and Software for Real Time Process Control. J. Zalewski, W. Ehrenberger (Eds.), pp. 555–563, North-Holland, Amsterdam, 1989

[11] IAEA: Manual on Quality Assurance for Computer Software. Final Version, International Atomic Energy Agency, Vienna, April 1987

[12] IEC: Application of Digital Computers to Nuclear Reactor Instrumentation and Control. IEC 643, International Electrotechnical Commission, Geneva, 1979

[13] IEC: Software for Computers in the Safety Systems of Nuclear Power Stations. IEC 880. International Electrotechnical Commission, Geneva, 1986

[14] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 729 1989

[15] IEEE Standard for Software Configuration Management Plans. IEEE Std 828 1983

[16] IEEE Standard for Software Test Documentation. IEEE Std 829 – 1983

[17] IEEE Standard for Software Unit Testing. IEEE Std 1008 – 1986

[18] IEEE Standard for Software Verification and Validation Plans. IEEE Std 1012 1986

[19] IEEE Standard for Software Reviews and Audits. IEEE Std 1028 – 1986

[20] IEEE Recommended Practice for Application of IEEE Std 828 to Nuclear Power Generating Stations. IEEE Std 1033 – 1985

[21] IEEE Guide to Software Configuration Management Planning. IEEE Std 1042 – 1987

[22] IEEE Guide for Software Verification and Validation Plans. IEEE Project 1059

[23] IEEE Standard for Software User Documentation. IEEE Std 1063 – 1986

[24] IEEE Standard for Software Life Cycle Processes. IEEE Project 1074

[25] Krebs H.: Verification of Safety Related Programs for a Maglev System. Working Paper 520, European Workshop on Industrial Computer Systems, TC 7, 21 July 1986

[26] Malagardis N.: Certification and Validation in Europe. Proc. Conf. Hardware and Software for Real Time Process Control. J. Zalewski, W. Ehrenberger (Eds.), pp. 545–553, North-Holland, Amsterdam, 1989

[27] Martin R. J., Osborne W. M.: Guidance on Software Maintenance. NBS Special Publication 500–106, National Bureau of Standards, Washington (DC), December 1983

[28] NBS: Guideline for Lifecycle Validation, Verification, and Testing of Computer Software. FIPS Publication 101, National Bureau of Standards, Washington (DC), June 1983

[29] Neumann A. J.: Management Guide for Software Documentation. NBS Special Publication 500-87, National Bureau of Standards, Washington (DC), January 1982

[30] Redmill F. (Ed.): Dependability of Critical Computer Systems. Vol 1. Elsevier Applied Science, London, 1988

[31] Redmill F. (Ed.): Dependability of Critical Computer Systems. Vol 2. Elsecier Applied Science, London, 1989

[32] Smith I. C.: The Integrity of Computer Based Safety Systems. Proc. Conf. Hardware and Software for Real Time Process Control. J. Zalewski, W. Ehrenberger (Eds.), pp. 163–173, North-Holland, Amsterdam, 1989

[33] Updike-Rumley M.: IEEE Standardization of Configuration Management Issues. Proc. Conf. Hardware and Software for Real Time Process Control J. Zalewski, W. Ehrenberger (Eds.), pp. 537–544, North-Holland, Amsterdam, 1989

[34] Zalewski J.: Dokumentacja oprogramowania – terminologia. Informatyka, nr 3, 5, 8, 10, 1984.

ZAKŁAD KOMPUTEROWYCH SYSTEMÓW POMIAROWYCH

ul.Zb. z Bogdańca 4
80-419 Gdańsk

digimer

tel. 419519

Oferuje usługi w zakresie:

- automatyzacji pomiarów, zbierania i przetwarzania danych pomiarowych w oparciu o komputery PC XT/AT,
- konstrukcji testerów produkcyjnych dla urządzeń elektronicznych cyfrowych i analogowych,
- realizacji interfejsów pomiarowych z oprogramowaniem,
- dostaw sprzętu komputerowego.

Produkujemy wyposażenie PC XT/AT z oprogramowaniem:

* PROGRAMATORY, SYMULATORY PAMIĘCI EPROM (16kb-2Mb)
– sprawdzone i ocenione pozytywnie przez 100 użytkowników w kraju i za granicą – symulatory 8/16bit

* TESTERY CYFROWYCH UKŁADÓW SCALONYCH
– dla TTL, CMOS ser 40xx i 45xx DRAM, SRAM – identyfikuje nieoznaczone układy – edytor oraz debugger wzorca testu

* WIELOKANAŁOWE PRZETWORNIKI A/D 12bit od 20μsec
– z separacją galwaniczną, preferowane do zastosowań w przemyśle, medycynie i na stanowiskach badawczych,
– gwarantowana klasa pomiarowa – software i aplikacja

* KARTY PROTOTYPOWE PC/XT (dł. 32,5 cm)

Dokumentacja. Gwarancja i serwis pogwarancyjny.

* EMULATOR 8031/32/35/48/51/52

0/2/90

INTERSOFT

00–496 Warszawa, ul. Krucza 16/22
Tel. 28-29-53

Szanowni Państwo!

Polecamy duży wybór dokumentacji dot. komputerów IBM oraz oprogramowania, m in.:

Przewodnik programisty IBM	144 000 zł	Turbo Power Tools Plus (procedury do TP 4/5)	115 000 zł
Wprowadzenie do komputerów IBM	32 000 zł	Grafika TP v.4.0 i TC v.1.5	110 000 zł
Sidekick	50 000 zł	Clipper 87, kompil. do dBase III+	160 000 zł
Wstęp do grafiki (Basic, Turbo, Graphics)	60 000 zł	dBase III+	170 000 zł
Autocad v.2.17	86 000 zł	dBase III+, praca w sieci	40 000 zł
Lotus 1–2–3 v.2.0	100 000 zł	Fox Base+	140 000 zł
Framework IIp	114 000 zł	Programowanie w assemblerze	160 000 zł
System operacyjny DOS 4.0	160 000 zł	Eureka	65 000 zł
System operacyjny DOS 3.3	144 000 zł	Poly-Windows	48 000 zł
Turbo Basic	130 000 zł	Instrukcja obsługi PC 1512	150 000 zł
Turbo „C" v.2.0	145 000 zł	Wordstar 2000	50 000 zł
Aztec „C"	132 000 zł	Modula 2 Logitech	102 000 zł
Zastosowanie języka C dla zaawansowanych	160 000 zł	Informix	180 000 zł
Turbo Graphics (do TP v.3.0)	74 000 zł	OS-2, opis systemu	130 000 zł
Turbo Pascal v.4.0	43 000 zł	Or-Cad	170 000 zł
Turbo Pascal v.5.0	270 000 zł	Novell, podr. użytkownika i instalatora	180 000 zł
Turbo Pascal v.5.5	306 000 zł		

Do większości pozycji dołączamy dyskietki z przykładami (koszt nośnika nie jest ujęty w wyżej wymienionych cenach).

Prowadzimy sprzedaż wszelkiego sprzętu komputerowego po bardzo konkurencyjnych cenach.

Do sprzedawanego przez nas sprzętu dodajemy bezpłatnie pakiety dokumentacji i oprogramowania o wartości zależnej od wielkości sprzedaży.

ZAPRASZAMY!

EO/232/89

LECH BANACHOWSKI
JAROSŁAW DEMINET
MAREK J. LAO
Instytut Informatyki
Uniwersytet Warszawski

Propozycja nowego programu uniwersyteckich studiów informatycznych (3)

Trzecia część artykułu jest poświęcona omówieniu programu wybranych przedmiotów bloku zaawansowanego. Podano tu jedynie hasła, które mogą stanowić podstawę programu omawianych przedmiotów. Dokładniejsze programy muszą być ustalone przez specjalistów oraz powinny uwzględnić najnowszą wiedzę. Wiedza bardzo specjalistyczna może być również przekazywana w wykładach monograficznych.

INF 16: Metody kompilacji

Dwa wykłady semestralne (być może musi to być niepodzielny wykład roczny). Dodatkowo (dla specjalistów) pracownia programowania po pierwszym z tych semestrów.

INF 16a: Metody kompilacji I (30+30 godz., egzamin)

Program wykładu
- Automatyczne generowanie analizatorów leksykalnych i składniowych.
- Metody półautomatycznej korekcji błędów przy analizie składniowej.
- Generowanie kodu i optymalizacja lokalna – usuwanie wspólnych podwyrażeń, optymalizacja przydziału rejestrów.
- Wynikowe struktury danych dla różnych konstrukcji programowych (wybrane).

INF 16b: Metody kompilacji II (30+30 godz., egzamin)

Program wykładu
- Analiza przepływu (zmienne żywe itp.).
- Optymalizacja globalna (międzyproceduralna).
- Optymalizacja szparkowa (peephole).
- Wynikowe struktury danych dla różnych konstrukcji programowych (zmienne dynamiczne – odśmiecanie, prefiksowanie itp.).
- Metody tłumaczenia języków obiektowych.
- Metody tłumaczenia języków z wiązaniem dynamicznym.

INF 16P: Metody kompilacji – pracownia programowania

Program pracowni
- Uruchamianie analizatorów (z użyciem takich programów, jak *lex* czy *yacc*).
- Uruchamianie makiet optymalizatorów dla ustalonej postaci pośredniej kodu (zależnie od tego, czy optymalizacja występuje w pierwszym semestrze).

INF 17: Semantyka języków programowania (30+30 godz., egzamin).

Dodatkowo (dla specjalistów) pracownia programowania po wykładzie.
Program wykładu
- Ogólne informacje o metodach definiowania semantyki: maszyna, język, realizacja, realizacja bezpośrednia języka programowania, metoda operacyjna, metoda algebraiczna z punktami stałymi, metoda denotacyjna, metoda aksjomatyczna (poszczególne metody powinny być ilustrowane semantyką jednego małego języka programowania).
- Semantyka wybranych konstrukcji; wyrażenia, instrukcje, struktura blokowa, sterowanie (kontynuacje), procedury, procedury rekurencyjne, przekazywanie parametrów, typy danych (najrozsądniej przedstawiać semantykę tych konstrukcji za pomocą opisu denotacyjnego).
- Porównanie metod: operacyjnej, denotacyjnej, aksjomatycznej.
- Zastosowanie gramatyk atrybutowych do definiowania semantyki.
- Problemy związane z definiowaniem semantyki współbieżności.

INF 17P: Semantyka języków programowania – pracownia programowania

Podczas zajęć studenci powinni wprowadzać konstrukcje z jednego do drugiego języka (np. rozszerzając systemy Lispu o nowe konstrukcje programowe). Można też korzystając z gotowych systemów uruchamiania prototypowych interpreterów dla opisu denotacyjnego (np. SPS, SIS) definiować własne języki.

INF18: Logiki programów (30+30 godz., egzamin)

Program wykładu
- Język i logika: język programowania i elementarne pojęcia logiki matematycznej.
- Składnia i semantyka logiki pierwszego rzędu: składnia – symbole, termy, formuły, semantyka – modele, interpretacje, prawdziwość w logice pierwszego rzędu.
- Własność logiki pierwszego rzędu: zmienne wolne i związane, podstawienie (zastąpienie), kolizje, przemianowanie, postać normalna.
- Rozumowanie logiczne: znaczenie semantyczne wnioskowania, składnia wnioskowania, system aksjomatyczny dla logiki pierwszego rzędu, reguły wnioskowania, niesprzeczność.
- Zupełność i zagadnienia rozstrzygalności: konstrukcja modeli, uniwersum Harbranda, twierdzenie Lindenbauma, twierdzenie o zupełności i niektóre wnioski z niego, logika z równością, problemy rozstrzygalności, złożoność, wnioski dotyczące własności stopu.
- Logiki programów: logika Floyda-Hoare'a, logiki modalne, logika algorytmiczna, logika dynamiczna.
- Aksjomatyczna metoda definiowania semantyki programów.

INF 18: Teoria obliczeń (30+30 godz., egzamin)

Program wykładu
- Uzupełnienie wiadomości o językach regularnych (automaty dwukierunkowe, hierarchia „głowicowa").
- Algorytmy wyszukiwania wzorca.
- Uzupełnienie wiadomości o językach bezkontekstowych (automaty dwukierunkowe, hierarchia głowicowa).
- Nierozstrzygalne problemy gramatyk bezkontekstowych.
- Deterministyczne języki bezkontekstowe (postać LR(k), twierdzenie Knutha).
- Hierarchie założoności (twierdzenie o hierarchi czasowej, pamięciowej, twierdzenie Savitcha).
- Pojęcie zupełności (przykłady problemów zupełnych w kilku klasach).
- Klasa NP (twierdzenie Cooka, twierdzenie Mahaney, problem PRIMES).
- Teoria Bluma (twierdzenie o przyspieszaniu, o luce, o sumie).
- Alternujące maszyny ze stosem (twierdzenie Cooka).

INF 21: Analiza algorytmów

Dwa wykłady semestralne. Dodatkowo (dla specjalistów) możliwe pracownie programowania po każdym semestrze lub równolegle z wykładami.

INF 21a: Analiza algorytmów I (30+30 godz., egzamin)

Program wykładu
- Pojęcie kosztu zamortyzowanego (przykłady),
- Samoorganizujące się struktury danych.
- Dolne ograniczenia dla wybranych problemów.
- Poprawianie złożoności przez dobór właściwej struktury danych.
- Algorytmy równoległe (model PRAM i modele sieciowe, zależności między różnymi modelami).
- Podstawowe techniki obliczeń równoległych (metoda cyklu Eulera, obliczenia na drzewach).
- Sortowanie równoległe.
- Problem *„list ranking„*.

INF 21P: Analiza algorytmu – pracownia programowania

Pracownia powinna być poświęcona realizacji poznawanych algorytmów w wybranym języku programowania, badaniu ich kosztu, eksperymentów probabilistycznych itp.

INF 25: Sztuczna inteligencja

Dwa wykłady semestralne, z których pierwszy powinien stanowić zamkniętą całość wprowadzającą do zagadnień sztucznej inteligencji. Dodatkowo (dla

specjalistów) pracownia programowa po zaliczeniu pierwszego semestru lub pracownia roczna równoległa z wykładem.

INF 25a: Sztuczna inteligencja I (30+30 godz., egzamin)

Program wykładu
- Wprowadzenie do sztucznej inteligencji: symulowanie tzw. inteligentnego zachowania w różnych dziedzinach, rozwiązywanie problemów, gry, języki naturalne, odpowiadanie na pytania, nauczanie itp., algorytmy heurystyczne a algorytmy gwarantujące rozwiązanie.
- Przykłady algorytmów heurystycznych w grach i rozwiązywaniu problemów.
- Systemy wspomagające dowodzenie twierdzeń, metoda rezolucji.
- Rozumienie języków naturalnych: techniki parsowania, języki bezkontekstowe i gramatyki transformacyjne, gramatyki Fillmore'a, zależności pojęciowe Shanka, analizatory, generowanie zdań, translacja.
- Reprezentacja wiedzy: rachunek pierwszego rzędu, klauzule Horna, sieci semantyczne, reguły produkcji, baza wiedzy, system wnioskowania, dedukcja, rezolucja.
- Języki sztucznej inteligencji: Lisp, Scheme, Prolog.

INF 25b: Sztuczna inteligencja II (30+30 godz., egzamin)

Program wykładu
- Systemy doradcze: istniejące systemy (DENDRAL, MYCIN,) metawiedza, przepływ wiedzy eksperta, systemy samoobjaśniające się.
- Rozpoznawanie wzorców: opisy strukturalne, opisy symboliczne, odbiór maszynowy, interpretacja, semantyka i modele, identyfikacja obiektów, rozpoznawanie mowy.

INF 25P: Sztuczna inteligencja – pracownia programowania

Pracownia powinna obejmować uruchomienie projektu z zakresu sztucznej inteligencji, np. teksty ilorazu inteligencji, rozpoznawania wzorców, wnioskowania (podsystemy doradcze).

INF 26: Modelowanie i symulacja (30+30 godz., egzamin)

Proponuje się dodatkowy wykład z rachunku prawdopodobieństwa i statystyki. Dodatkowo (dla specjalistów) pracownia programowania po wykładzie.
Program wykładu
- Modelowanie: różne rodzaje modeli, własności modeli liniowych, techniki konstruowania modeli, podejście „blackbox", rola modelu w badaniach, poprawność, parametry pierwszego i drugiego rzędu.
- Metodologia symulacji: metodologia badań symulacyjnych korzystanie z właściwego narzędzia-języka (np. GPSS, SIMSCRIPT).
- Generowanie liczb losowych: generatory liczb pseudolosowych, prze kształcenia liczb losowych, testy losowości (wiarygodności).
- Oszacowania parametrów: maksymalna zgodność (likelyhood), najmniejszy średni błąd kwadratowy.
- Projekt eksperymentu: faktoryzacja, optymalizacja.

INF 26P : Modelowanie i symulacja – pracownia programowania

Podczas zajęć pracowni programowania studenci powinni zastosować przedstawiane techniki, narzędzia i metody z wykładu do konkretnego zadania symulacyjnego.

INF 29: Organizacja pracy zespołowej i pracy ośrodka obliczeniowego (30+30 godz., egzamin)

Wykład powinien być w miarę możliwości prowadzony przez informatyka-praktyka (osobę z zewnątrz). Dodatkowo (dla specjalistów) pracownia programowania równolegle z wykładem lub po jego zakończeniu.
Program wykładu
- Znaczenie standardów.
- Organizacja
- Standardy weryfikacji projektu.
- Standardy projektów zespołowych: organizacja zespołu, zadania, oszacowanie liczebności zespołu i odpowiedzialności, opis zadań, wymagania szkoleniowe, pomocnicze standardy, bezpieczeństwo, *back-up* i odzyskiwanie, określenie środowiska, sprzęt i oprogramowanie, usługi z zewnątrz, standardy sprawozdań, zadania bibliotekarza projektu.
- Dokumentacja, standardy techniczne, cykl życia projektu.
- Administracja ośrodka obliczeniowego.
- Personel ośrodka.
- Sprzęt: kierunki zmian w rozwoju.
- Wybór komputera i ocena wydajności sprzętu.
- Równoważenie sprzętu i personelu.
- Ocena oprogramowania i jego wybór.

INF 29P: Organizacja pracy zespołowej i pracy ośrodka obliczeniowego – pracownia programowania

Pracownia powinna polegać na pełnej realizacji projektu zgodnie z zasadami podanymi w wykładzie w zespołach od 5 do 8 osób. Projekt powinien uwzględniać wymagania dokładnej specyfikacji powiązań.

INF 30: Kombinatoryka (30+30 godz., egzamin)

Program wykładu
- Funkcje tworzące i ich własności.
- Wykładnicze funkcje tworzące.
- Formuły inwersyjne.
- Równania rekurencyjne.
- Pojęcie matroidu – algorytmy zachłanne.
- Pojęcia i zastosowania wielomianowej metody operatorowej.
- Kombinatoryczna teoria zliczania (twierdzenia Poly'a i Burnside'a).
- Kombinatoryczne metody kodowania.
- Metoda RSA (Rivesta, Shamira, Adlemana) bezpiecznego kodowania.
- Probabilistyczny algorytm Strassena badania czy liczba jest pierwsza.

INF 31: Teoria grafów (30+30 godz., egzamin)

Proponuje się dodatkowo pracownię programowania.
Program wykładu
- Usystematyzowanie wcześniej poznanych pojęć.
- Kolorowanie grafów (twierdzenie Brooksa).
- Grafy płaskie (twierdzenie Kuratowskiego) i ich kolorowanie.
- Grafy zewnętrzne planarne i ich kolorowanie.
- Pojęcie cyklu Hamiltona i warunki dostateczne na stwierdzenie istnienia (analiza probabilistyczna).
- Przepływy i wnioski z twierdzenia Forda-Fulkersona (twierdzenie Mengera, Halla, twierdzenie węgierskie).
- Grafowe problemy NP–zupełne.
- Algorytmy aproksymacyjne.
- Metody hierarchicznego opisu grafów.
- Pojęcie *expandera*, własności i metody konstruowania.

INF 35: Zastosowania informatyki (30+30 godz., egzamin)

Wskazany duży udział specjalistów z zewnątrz.
Program wykładu
- Systemy komputerowe: praca wsadowa i interakcyjna, praca w czasie rzeczywistym, zarządzanie informacją, sieci; opis każdego z tych systemów, różnica, rodzaje zastosowań.
- Bazy danych: zakładanie i użytkowanie, definicje danych i struktur.
- Błędy: typy, skutki, postępowanie w razie powstania błędów.
- Skutki społeczne: sposoby współdziałania człowieka z komputerem, ochrona danych osobowych, problemy moralne i prawne.
- Przyszłe skutki społeczne: społeczeństwo bez gotówki i czeków, nauczanie wspomagane przez komputer, narodowe banki danych.
- Języki: wprowadzenie (stosownie do potrzeb) w specyfikę języków ukierunkowanych na przetwarzanie danych administracyjnych, języków przetwarzania napisów, krótka charakterystyka cech języków przydatnych dla konkretnych grup problemów.
- Wybrane pojęcia i metody stosowane przy rozwiązywaniu problemów na konkretnych przykładach.

Przykładowe specjalizacje

Począwszy od VI semestru wybór odpowiednich zestawów wykładów i zajęć powinien zapewnić uzyskanie przez absolwentów następujących przykładowych specjalizacji.

Programista systemowy (ukierunkowany na systemy operacyjne)

Semestr VI (14 godz. tygodniowo, 2 egzaminy)	
INF12: Projekt programistyczny	(2)
INF13: Podstawy elektroniki...	(4)
INF14a: Systemy operacyjne I	(4) E
INF16a: Metody kompilacji I	(4) E
Semestr VII (16 godz. tygodniowo, 3 egzaminy)	
seminarium	(2)
wykład monograficzny	(2)
INF14b: Systemy operacyjne II	(4) E
INF34: Teoria kolejek	(4) E
INF21a: Analiza algorytmów I	(4) E
Semestr VIII (15 godzin tygodniowo, 2 egzaminy)	
seminarium	(2)
wykład monograficzny	(2)
INF14P: Pracownia systemów operacyjnych	(3)
INF27: Systemy mikrokomputerowe	(4) E
INF15: Wydajność systemów operacyjnych	(4) E
Semestr IX (13 godzin tygodniowo, 1 egzamin)	
seminarium	(2)
wykład monograficzny	(2)
INF27P: Pracownia systemów mikrokomputerowych	(3)
INF35: Zastosowania informatyki	(2)
INF29: Organizacja pracy zespołowej	(4) E
Semestr X (magisterski: 4 godz. tygodniowo)	
seminarium	(2)
wykład monograficzny...	(2)
praca magisterska	

Teoretyk (ukierunkowany na teorię programowania)

Semestr VI (14 godz. tygodniowo, 2 egzaminy)
- INF13: Projekt programistyczny (2)
- INF13: Podstawy elektroniki... (4)
- INF18: Logiki programowania (4) E
- INF16a: Metody kompilacji I (4) E

Semestr VII (16 godz. tygodniowo, 3 egzaminy)
- seminarium (2)
- wykład monograficzny (2)
- INF16b: Metody kompilacji II (4) E
- INF17: Semantyka języków programowania (4) E
- INF21a: Analiza algorytmów I (4) E

Semestr VIII (15 godz. tygodniowo, 2 egzaminy)
- seminarium (2)
- wykład monograficzny (2)
- INF17P: Pracownia do semantyki (3)
- INF19: Teoria obliczeń (4) E
- INF21b: Analiza algorytmów II (4) E

Semestr IX (13 godz. tygodniowo, 1 egzamin)
- seminarium (2)
- wykład monograficzny (2)
- INF21P: Pracownia do analizy algorytmów (3)
- INF35: Zastosowania informatyki (2)
- INF25a: Sztuczna inteligencja (4) E

Semestr X (magisterski: 4 godz. tygodniowo)
- seminarium (2)
- wykład monograficzny... (2)
- praca magisterska

Teoretyk (ukierunkowany na analizę algorytmów)

Semestr VI (14 godz. tygodniowo, 2 egzaminy)
- INF12: Projekt programistyczny (2)
- INF13: Podstawy elektroniki... (4)
- INF21a: Analiza algorytmów I (4) E
- INF20: Teoria złożoności (4) E

Semestr VII (15 godz. tygodniowo, 2 egzaminy)
- seminarium (2)
- wykład monograficzny (2)
- INF21b: Analiza algorytmów II (4) E
- INF33: Równania różniczkowe (4) E
- INF21P: Pracownia do analizy algorytmów (3)

Semestr VIII (14 godz. tygodniowo, 2 egzaminy)
- seminarium (2)
- wykład monograficzny (2)
- INF22: Optymalizacja (4) E
- INF30: Kombinatoryka (4) E
- INF35: Zastosowania informatyki (2)

Semestr IX (13 godz. tygodniowo, 1 egzamin)
- seminarium (2)
- wykład monograficzny (2)
- INF22P: Pracownia do optymalizacji (3)
- seminarium matematyczne (2)
- INF19: Teoria obliczeń (4) E

Semestr X (magisterski: 4 godz. tygodniowo)
- seminarium (2)
- wykład monograficzny (2)
- praca magisterska

Autorzy oczekują zgłaszania uwag oraz propozycji uzupełnień lub zmian do przedstawionego programu studiów.

Atrybuty jakości oprogramowania – definicje podręcznikowe

dokończenie z III s. okładki

Współdziałanie (ang. *interoperability*) – miara wysiłku wymaganego do zespolenia jednego systemu z innym.

Zupełność (ang. *completeness*) – pełne zaimplementowanie wymaganych funkcji.

Zwięzłość (ang. *conciseness*) – zaimplementowanie funkcji przy użyciu minimalnej długości kodu.

JANUSZ ZALEWSKI

Micomp

ZAKŁAD SYSTEMÓW MIKROKOMPUTEROWYCH
MICOMP

40–045 Katowice, ul. Astrów 7
telefon i telefaks: 518-628
teleks: 315687

Systemy teletransmisji danych
ICL 1900, 2900, ME 29, 39,
ODRA 1305, EMCX 1305:

skaner MPXSCAN-8007,
procesor sieci WAN
MICOMP-8075
(emulacja ICL 7503),
program teletransmisji danych
MICROS FXBM
(PC, rozproszone bazy danych,
wersja sieciowa – NOVELL),
adaptery, modemy, testery,
emulatory terminali

Integracja systemów ICL i ODRA
z mikrokomputerami standardu PC
i systemami NOVELL, UNIX SCO

Instalacje w największych systemach
na terenie całego kraju!

EO/1124/89

Symboliczny program uruchomieniowy dla Ady (1)

Centralnym pojęciem języka Ada, podobnie jak jego poprzedników, jest typ pozwalający na definiowanie reprezentacji obiektów poszczególnych rodzajów i zbioru dozwolonych na nich operacji. Notacja podtypów jest wygodnym środkiem umożliwiającym weryfikację poprawności operacji przypisania. Doświadczenie wykazało, że weryfikacja taka jest przydatna, chociaż nie wiąże się bezpośrednio z semantyką poszczególnych programów.

Sprawdzenie poprawności programu pod względem spójności (franc. *coherence*) może odbywać się już w fazie kompilacji przy założeniu, że niespójności wykrywane w czasie wykonywania programu są obsługiwane za pomocą mechanizmu sytuacji wyjątkowych. Jeżeli taka weryfikacja okaże się kompletna, to diagnostyka sytuacji wyjątkowych będzie wystarczająca do wykrycia błędów programu.

Niektóre sytuacje błędne nie są jednak wykrywalne przez kompilatory Ady. Często zdarza się również, że program spójny zachowuje się niezgodnie z założeniami z powodu błędnego jego zakodowania. Wreszcie niektóre sytuacje anormalne nie są uważane przez kompilator Ady za sytuacje błędne. Dlatego też istnieje potrzeba użycia narzędzia umożliwiającego śledzenie zachowania programu podczas jego wykonywania. Takie narzędzie wykrywające sytuacje błędne podczas wykonywania programu może być traktowane jako naturalne przedłużenie kompilatora pozwalające na uzyskanie efektu odpowiadającego wprowadzeniu dodatkowych podtypów w języku.

ŹRÓDŁA BŁĘDÓW W PROGRAMACH ADY

Poniżej przedstawiono podział wszystkich błędów występujących w Adzie.

Niewyspecyfikowane sytuacje wyjątkowe

Błąd wykryty przez program może nie posiadać odpowiadającej mu procedury obsługi. Oznacza to, że albo jest to sytuacja anormalna, która nie powinna zajść, albo brakuje procedury jej obsługi. Taka sytuacja wywołuje zakłócenie wykonywania zadania, często bez jakiegokolwiek komunikatu o błędzie.

Sytuacje błędne

Jest to szczególna forma poprzedniego przypadku dla sytuacji wyjątkowej **PROGRAM-ERROR.** Dokładna diagnostyka nie jest tu możliwa i kompilatory nie wykonują stosownych testów. Zwykle argumenty operacji są testowane dynamicznie, co na ogół zapewnia skuteczne wykrywanie błędów. Podstawową funkcją programu uruchomieniowego jest umożliwienie rozróżnienia poszczególnych błędów i miejsc ich powstania.

Podstawowe sytuacje błędne to:
- niezainicjowane zmienne,
- wielokrotne dostępy do zmiennych, jak: błędna modyfikacja wartości indeksu, uzależnienie od implementacji mechanizmu przekazywania parametrów do procedur,
- użycie nieokreślonych zasobów, np.:
  - próba dostępu do obiektu dealokowanego (**UNCHECKED_DEALLOCATION**),
  - niezgodność typów (**UNCHECKED_CONVERSION**),
  - nieprawidłowe użycie porównania (**for x'Address use**),
  - inne błędy wynikające z zaniechania testów systematycznych,
- korzystanie ze specyficznych własności kompilatora nie określonych przez normę języka (np. kolejność wartościowania wyrażeń),
- niesynchronizowany dostęp do zmiennych dzielonych.

Sytuacje niepożądane

Część sytuacji nie jest uważana za błędne, chociaż jest możliwe statyczne ich wykrycie. Należą do nich między innymi konstrukcje związane z zadaniami, np. zakleszczenie, lokalne lub globalne, nie jest sytuacją błędną i w związku z tym ani kompilator, ani system operacyjny nie przekaże komunikatu błędu. Jeżeli jedyną otwartą opcją instrukcji **select** zawartej wewnątrz konstrukcji **accept** jest **terminate**, to proces zablokuje się w sposób nieokreślony.

Pętle

Problem zapętlenia jest związany ze wszystkimi językami sekwencyjnymi. W Adzie ma on szczególne właściwości. Przyczyna zapętlenia jest zawsze lokalizowalna i prawie zawsze określona. Konsekwencje zapętleń są szczególnie groźne dla programów działających w czasie rzeczywistym, z powodu konieczności zaniechania ich pracy. Inne przypadki zapętleń, praktycznie nierozwiązywalne w innych językach programowania, w Adzie mogą spowodować pojawienie się sytuacji błędnych typu **STORAGE-ERROR** będących integralnymi elementami Ady. Program zapętleń pojawia się stosunkowo często przy uruchamianiu programów, pożądana jest więc łatwość wykrywania pętli w programach.

FUNKCJA OBSŁUGI BŁĘDÓW

Podstawową funkcją programu uruchomieniowego jest umożliwienie wykrycia (przez testowanie) sytuacji anormalnych, mniej lub bardziej skomplikowanych, i odpowiednia reakcja. Ogólna jej postać jest następująca:

(Kontekst:) zasięg zdarzenia (⇒ akcja)

Zasięg testu może być lokalny lub globalny. Jeżeli test jest globalny, to zdarzenie jest możliwe do lokalizacji we wszystkich kontekstach. Zasięg lokalny ogranicza wykonywanie testu do określonego kontekstu. Kontekst jest rozszerzeniem pojęcia ramy (ang. *frame*). Określa go blok, ciało podprogramu, ciało typu zadaniowego, ciało pakietu – wszystko, co według normy Ady może zawierać strefę wyjątków. Ponadto jest określony przez zbiór związanych z nim obiektów, w ogólności cały jego obraz lub dowolny podzbiór. **Zdarzenie** jest rozszerzeniem pojęcia wyjątku. Pozwala ono opisać każdą sytuację statyczną lub dynamiczną niezależnie od stopnia jej złożoności. Ogólnie – jest to opis statycznej lub dynamicznej sytuacji anormalnej zapowiadającej błąd. Kiedy zachodzi taka sytuacja mówimy o zajściu zdarzenia i wówczas jest wykonywana związana z nim klauzula obsługi. Akcja jest rozszerzeniem pojęcia instrukcji. Reasumując, podstawową funkcją programu jest przekazanie maksymalnej liczby informacji związanych z obsługiwanym zdarzeniem tak, aby ułatwić diagnostykę błędu lub znaleźć przyczynę jego powstania.

ZDARZENIE

Wykrywanie zdarzeń polega na wykonywaniu i analizowaniu testów, które omówiono poniżej.

Podstawowe testy

W najprostszym przypadku test może reagować na określoną wcześniej nazwę powstałej sytuacji wyjątkowej. Ma on wtedy postać:

when CONSTRAINT_ERROR ⇒ action

Taka forma przechwytywania sytuacji wyjątkowych programu jest równoważna dynamicznej definicji procedury obsługi sytuacji wyjątkowej. Najczęstszym sposobem reakcji na sytuacje błędne powstałe w czasie wykonania programu bez udziału mechanizmu sytuacji wyjątkowych jest użycie klauzuli **others**, umożliwiającej lokalne wychwycenie sytuacji błędnej w konkretnym kontekście.

Istotnym ograniczeniem powyższych metod opisu zdarzeń jest ich oparcie na definicji podtypów w Adzie, co jest niekiedy niewystarczające

do śledzenia poprawności i zachowania programu. Przyjęto więc, że zdarzenie może zostać wywołane przez ewaluację każdej relacji Ady na jednej lub wielu zmiennych. Przykładowo, definicja podtypu:

subtype int2 **is integer range** 10 .. 20;
Z : int2;

jest interpretowana jako wyrażenie

if Z **not in** 10 .. 20 ⇒ **raise** CONSTRAINT_ERROR.

Natomiast następujące testy nie mogą być specyfikowane w Adzie:
if Z **not in** O..K --jeżeli K jest zmienną,
if Z mod 2 >0 --Z ma być parzyste.

Test modyfikacji

Testowanie modyfikacji obiektu pozwala na wybiórcze, dokładne określenie wystąpienia sytuacji błędnej. Jest ono zapisywane w notacji programu uruchomieniowego jako **mod**(v) i może być wykonywane na dowolnych obiektach Ady. Celem takiego testu jest określenie źródeł modyfikacji wartości obiektu, również w środowisku wielozadaniowym, co pozwala na znalezienie fragmentu programu wykonującego błędne operacje na danym obiekcie. Przy pomocy takiego testu można również wykryć brak modyfikacji obiektu w miejscu, gdzie powinna ona nastąpić, np. brak przypisania parametru typu **out** w procedurze.

Test zapętleń

Test ten jest wykonywany w celu znalezienia zapętleń w programie. Pętla występująca w danym zadaniu jest zawarta wewnątrz określonej procedury. Mogą również istnieć pętle sprzęgające kilka zadań. Na skutek asymetrii mechanizmu spotkań zawsze jest możliwe wyróżnienie pętli czynnych i biernych. Na ogół przeprowadzanie dokładnego testu jest zbędne (ponieważ pętla może zakończyć się instrukcją **abort**), dlatego też testowana jest tylko droga wykonywania programu i na jej podstawie wnioskuje się o zapętleniach.

Test sekwencyjny

W niektórych zastosowaniach programu uruchomieniowego bywa wygodne zdefiniowanie zdarzenia jako faktu wykonania konkretnej instrukcji lub instrukcji określonego rodzaju. W celu wybrania grupy instrukcji można posługiwać się różnorodnymi kryteriami. W ten sposób testowaniu może podlegać wykonanie lub napotkanie w programie instrukcji wywołania podprogramu **return, accept, delay, goto, raise** itp.

Arytmetyka zdarzeń

Specyfikacja zdarzeń ma zadanie umożliwienia opisu sytuacji niepożądanych w programie, pozwalając programowi uruchomieniowemu na ich lokalizację. W bardziej skomplikowanych sytuacjach często jest konieczne tworzenie wyrażeń, będących dowolną kombinacją zdarzeń, za pomocą następujących operatorów:
or – unia,
and – koniunkcja,
then – sekwencja,
times – sekwencja o określonej długości, np. 100 **times mod** (x)
Operator **and** pozwala zdefiniować wiele nowych typów testów elementarnych. Operacja **then** pozwala opisywać sekwencje sytuacji występujących w różnych pakietach i zadaniach. W razie błędu znana jest wtedy sekwencja zdarzeń, które go poprzedzały. Możliwa jest także kombinacja zdarzeń z kontekstami ich występowania, co umożliwia opis błędów pośrednich, np. w pakietach, procedurach rekurencyjnych lub pracach wielozadaniowych. Są one określane jako

zdarzenie **in** kontekst

* * *

Możliwe jest też wyspecyfikowanie kilku zdarzeń asynchronicznych, które zapewniają sprzężenie z użytkownikiem, np.

Key_call: interwencja użytkownika;
task abort: porzucenie wykonywania bieżącego zadania.

Przykłady zdarzeń podano na załączonym wydruku.

```
when others                                -- klauzula wybierana w przypadku
                                           -- braku procedury obsługi danej
                                           -- sytuacji wyjątkowej;
if CONSTRAINT_ERROR and J>10               -- w przypadku wystąpienia sytua-
                                           -- cji błędnej CONSTRAINT_ERROR
                                           -- i gdy J > 10;
when NUMERIC_ERROR or CONSTRAINT_ERROR     -- treść oczywista
when I>10 then mod(A(I+5))                 -- jeżeli przypisanie I wartości
                                           -- większej niż 10 powoduje mody-
                                           -- fikację A(I+5);
when K times mod(J) and J/=K+3             -- jeśli po K-tej modyfikacji
                                           -- J jest różne od K+3;
when 10 times (call or return)             -- przy dziesiątym napotkaniu
                                           -- call lub return, z zachowaniem
                                           -- śladu programu;
when loop                                  -- znalezienie miejsca zapętlenia
P: if loop (N**3*5)                        -- złożoność P jest rzędu N^3;
if I**2 + J**2 > 100 at P.45               -- przy prawdziwości relacji w o-
                                           -- kreślonym miejscu programu;
```

Przykłady zapisu i znaczenia poszczególnych stwierdzeń

Oprac. **D. INKIELMAN**

Środowisko programowe...

dokończenie ze s. 12

Wykonanie płytki drukowanej również można bardzo przyspieszyć, zarówno na etapie projektowania, jak i konstrukcji, w wyniku automatyzacji obu procesów. Niebagatelną sprawą jest także możliwość szybkiego aktualizowania dokumentacji technicznej.

Opisane pakiety oprogramowania zostały przetestowane w trakcie wykonywania kilku projektów; wykazały ogromne walory użytkowe i potwierdziły swoją przydatność zwłaszcza w pracach naukowo-badawczych. Dzięki ich zastosowaniu można było szybko usuwać błędy, wprowadzać poprawki i optymalizować pakiety sprzętowe; powstały również warunki do równoległej realizacji kilku wariantów rozwiązań sprzętowych i wyboru najlepszego z nich (przy określonych ograniczeniach), w drodze symulacji i testowania rzeczywistego pakietu.

LITERATURA

[1] Dokumentacja techniczna pakietu Orcad
[2] Dokumentacja techniczna pakietu Redac
[3] Katalog firmy Quest.

Wychodząc naprzeciw zapotrzebowaniu środowiska, redakcja nasza oferuje łamy INFORMATYKI na bezpłatne zamieszczanie anonsów o imprezach informatycznych organizowanych na terenie całego kraju.

Zwracamy się do organizatorów konferencji, wystaw, konkursów i sympozjów z prośbą o przesyłanie do redakcji krótkich (pół strony maszynopisu) informacji o planowanych imprezach. Teksty powinny zawierać datę i miejsce imprezy, krótką charakterystykę jej tematyki oraz adres organizatorów.

Oczekujemy na zgłoszenia.

Nowa generacja sieci lokalnych

Projektanci sieci lokalnych mają do dyspozycji nowy program zarządzania tymi sieciami – OS/2 Lan Manager firmy Microsoft Corporation – działający pod nadzorem systemu operacyjnego OS/2. Jednocześnie ogłoszono pierwsze wyroby opracowane dla tego oprogramowania: 3+ Open firmy 3Com i Tapestry II firmy Torus. Przewiduje się (rys.), że w latach 1989–1990 zapotrzebowanie na OS/2 przekroczy zapotrzebowanie na DOS. Dzieje się tak dlatego, że OS/2 jest systemem otwartym i zawiera ponad 250 sprzęgów programowych (ang. *applications programmation interfaces*) pozwalających na rozliczne zastosowania. Ponadto jest to system wielozadaniowy i oferujący znacznie większą przestrzeń pamięciową.

Konkurentem programu OS/2 Lan Manager, który został opracowany we współpracy z 3Com, jest Lan Server firmy IBM, który w listopadzie 1988 r. wciąż jeszcze był w opracowaniu. Jest to tzw. oprogramowanie zarządzenia sieciami trzeciej generacji. Pierwsza generacja umożliwiała współdzielenie zasobów (drukarki, stacje dysków itd.). Bardziej rozbudowana druga generacja umożliwiała współdzielenie plików, ale jeszcze w sposób nierozproszony, gdzie zastosowanie traktowano na poziomie stanowiska wywołującego. Do tej kategorii należą Netware firmy Novell, 3+ 3Com i PC Lan Program.

LAN MANAGER

Przy wykorzystaniu OS/2 Lan Manager wygląda to inaczej. System potokowy (ang. *pipes*) wprowadzony w OS/2 umożliwia komunikację między zadaniami rozłożonymi w sieci. Na przykład, zarządzanie bazą danych SQL może być rozbite na dwa bloki. Pierwszy zawiera procedury zapytań do bazy, a drugi samo zarządzanie. W sieci zarządzanej przez Lan Manager pytania są formułowane na poziomie stacji, a przetwarzanie danych jest dokonywane w komputerze-usługodawcy (ang. *server*), zawierającym bazę. Pozwala to z jednej strony wykorzystać dużą moc obliczeniową usługodawcy, a z drugiej zwolnić magistralę łączącą sieć. Łatwiejsze zadania, jak zarządzanie sprzęgiem użytkownika, są realizowane w stacjach.

Obecnie protokołem używanym do sieci PC jest Netbios. TCP/IP jest przeznaczony dla środowiska Unix z możliwością dołączania do sieci PC dzięki sprzęgowi Netbios. W przyszłości obowiązywać będzie protokół uniwersalny zalecany przez ISO.

IBM	TCP/IP	ISO	Warstwy
SMB	SMB NFS	Ftam i in.	7
		ASN. 1	6
Netbios	Netbios		
Netbios IBM		Warstwy sesji ISO	5
	TCP	Warstwy transportowe ISO	4
	IP	X25	3
LLC Macintosh	Macintosh	LLC Macintosh	2
Ethernet Token Ring, Starlan	Ethernet	Ethernet Token Ring	1

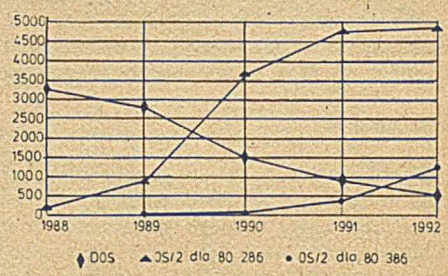
W praktyce Lan Manager składa się z trzech modułów: usługodawcy, części zapytującej i protokołów. Stacja usługodawcy może zawierać wszystkie trzy moduły dzięki wielozadaniowemu charakterowi OS/2. Dowolna stacja sieci obejmuje dwa ostatnie moduły. Zapowiedziana na początek 1989 r. wersja 1.1 OS/2 DBDC (ang. *Data Base Data Communications* – komunikacja danych bazy danych) firmy IBM zawiera część zapytującą. Komunikację między usługodawcą a częścią zapytującą obsługuje protokół SMB (ang. *Server Message Block* – blok komunikatów usługodawcy). Protokół ten opracowany początkowo przez firmę Microsoft, do sieci MS-Net, został przyjęty przez IBM do PC Lan Program i stanowi dziś standard w tej dziedzinie. Jego wykorzystanie pozwala włączyć do sieci w roli usługodawców komputery inne niż PC i PS, na przykład pracujące pod nadzorem Unixa lub komputery firmy DEC, na których zainstalowano SMB.

Realizacja warstw od 3 do 5 modelu OS/ISO (tabela) jest oparta na Netbiosie. Wybór kabli i metody dostępu (a więc sieci Ethernet, Token Ring lub Starlan) jest dowolny, o ile zgodny z Netbiosem. Obecnie, aby dostarczyć gotowy system użytkownikowi wytwórca musi opracować własny Netbios, lecz na początek 1989 r. przewidywano dostarczanie wraz z programem Lan Manager obsługi protokołów TCP/IP i zalecanych przez ISO.

Dzięki dostosowaniu do standardów rynkowych (SMB, Netbios, OS/2) i dobrej pracy (konstruktorzy uzyskują rezultaty testów zbliżone do Netware dla sieci zamkniętej i lepsze przy połączeniach z innymi sieciami), a także dzięki otwartości (Lan Manager zawiera ponad 150 sprzęgów API) system ten powinien zainteresować szeroką rzeszę konstruktorów i twórców systemów informatycznych. Ostatnio DEC przystosował Lan Managera do komputerów VAX. Innym obszarem zastosowań jest środowisko Unix. Około trzydziestu przedsiębiorstw (m.in. Apricot, Bull Goupil, HP, Nokia, Ungermann-Bass) podpisało kontrakty licząc na opracowanie wyrobów związanych z Lan Managerem.

TAPESTRY

Torus jest powstałą w 1983 r. firmą brytyjską, którą we Francji reprezentuje Lincs. Specjalizuje się ona w informatyce biurowej. Oferuje, między innymi, karty sieci Ethernet do IBM PC i PS, ale standardowym jej wyrobem jest oprogramowanie do zarządzania sieciami o nazwie Tapestry. Pierwsza wersja powstała w 1984 r., a firma Lincs przygotowała wersję francuską. Kolejna odmiana, Tapestry II, obejmuje Lan Manager i będzie służyć w przyszłości sieciom OS/2. Obecnie dostępna wersja działa jeszcze pod nadzorem DOS-a, przy czym usługodawcą może być IBM PC. Wersja z usługodawcą OS/2 była przewidywana na kwiecień 1989 r. Można wykorzystywać to oprogramowanie w stacjach OS/2, tworząc sieci do sześciu użytkowników.



Rozwój systemów operacyjnych DOS i OS/2 – dostawy w tysiącach sztuk (Źródło: Forrester Research)

Tapestry II zwiększa liczbę użytkowników sieci do 60 tysięcy, każdy usługodawca może wykorzystywać sześć stacji dyskowych i siedem drukarek; jest też możliwość dołączenia do sieci X25, co umożliwia dołączenie odległych stacji. Usługi biurowe systemu to poczta elektroniczna, kalendarz, a także możliwość podziału na podsieci. Drugie z wymienionych opracowań, 3+ Open firmy 3Com, również oparte na Lan Managerze, jest bardziej zaawansowane i pozwala już obecnie tworzyć sieci pod nadzorem systemu OS/2. Oprócz podziału urządzeń zewnętrznych (złożony system przydziału drukarek z możliwością oferowania kilku urządzeń tej samej kolejce), sprzęgu z użytkownikiem w postaci menu zgodnego z SAA (ang. *Systems Application Architecture* – architektura zastosowania systemów firmy IBM), myszy i adapterów konwersacyjnych, 3+ Open zapewnia wiele funkcji specjalnych. Można tu sterować usługodawcą zdalnie, za pomocą jednej ze stacji, można też, uwalniając zasoby przetwarzania, powierzać mu zadania niesieciowe.

3+ OPEN

Znane są już dwa moduły zastosowań współpracujących z 3+ Open. Pierwszy z nich, Lan View, pozwala na analizę zdarzeń zachodzących w sieci. Wykorzystuje on program *Presentation Manager*, który jest odpowiednikiem programu *Windows* dla systemu OS/2 i dostarcza obraz graficzny statystyk usługodawcy dotyczących wydajności, wykorzystania zasobów, ochrony itp. Drugi moduł, Lan Secure, zajmuje się przede wszystkim tym ostatnim zagadnieniem. Miały one być rozpo-

wszechniane w 1989 r. Różne usługi dodatkowe, jak poczta elektroniczna, zarządzanie terminarzami, możliwość dołączenia do SNA (ang. *Systems Network Architecture* – architektura sieci systemów firmy IBM) przewidywano w najbliższych miesiącach. Opracowano dwie wersje 3+ Open. Pierwsza umożliwia tworzenie sieci do pięciu użytkowników. Usługodawca powinien mieć 3 MB pamięci operacyjnej i 7 MB na dysku stałym. Koszt takiej konfiguracji wynosi ok. 1500 dolarów. Druga wersja nie ogranicza liczby użytkowników, a pojemność pamięci operacyjnej usługodawcy wynosi 4 MB. Wersja ta jest trzykrotnie droższa.

O ile usługodawca musi pracować pod nadzorem systemu OS/2, to dla stacji sieci wystarcza DOS. Pod koniec 1989 r. będzie możliwe dołączenie komputerą Macintosh. Stacje sieciowe muszą mieć co najmniej 512 KB pamięci operacyjnej przy pracy z DOS-em (zaleca się 640 KB) i 2 MB przy pracy z OS/2 niezależnie od przyjętej wersji sieci. Oprogramowanie to jest zgodne z 3+ (zwłaszcza z jego podstawowym elementem 3+ Share) i opiera się na protokole komunikacyjnym XNS (ang. Xerox Network System – system sieci Xerox). Dla 3+ Open osiągnięto zgodność z Netbios i TCP/IP, a na początku 1989 r. dotyczyć to miało protokołów warstw od 3 do 5 OSI/ISO.

Do połączeń z Ethernet firma 3Com zaleca wykorzystanie swych kart Etherlink (zwykła, Plus, II i MC), a do Token Ring karty Tokenlink (zwykła i Plus). Są one zgodne z XNS, lecz Netbios nie działa z wersją Plus.

Opracował J. R.
na podstawie Minis et Micros

Tranzystory kwantowe

Chociaż do praktycznego zastosowania półprzewodników z efektem kwantowym pozostało jeszcze kilka lat, ich właściwości zapowiadają się tak interesująco, że warto już dziś się z nimi zapoznać.

Podstawową cechą tych tranzystorów jest szybkość propagacji ponad tysiąckrotnie większa niż w tranzystorze klasycznym, ponadto wymiary ich są około stukrotnie mniejsze niż zwykłych tranzystorów. Działają one na zupełnie innych zasadach, znanych pod nazwą efektów kwantowych, które rządzą zachowaniem się materii i energii przy wymiarach rzędu 0,02 μm lub mniej. Przypomnijmy, iż teoria kwantowa twierdzi, że energia promieniująca objawia się, jak materia, przez strukturę nieciągłą – istnieje w postaci ziaren lub kwantów, dających się w rezultacie opanować.

W ten sposób można stworzyć bipolarny tranzystor z tunelem rezonansowym, będący pierwszym elementem zdolnym do sterowania „studnią kwantową" – ultracienką warstwą, która pozwala na przejście tylko elektronów naładowanych do odpowiednich energii.

Dyrektor techniczny firmy Texas Instruments – G. Heilmeier – powiedział, że przyszłe układy złożone z tych tranzystorów będą mogły spełniać stokrotnie więcej funkcji przy tych samych wymiarach zewnętrznych, a pobór mocy będzie znacznie mniejszy niż układów istniejących obecnie. I chociaż termin ich wprowadzania można ocenić na dziesięciolecie, oznaczają one możliwość realizacji superkomputera przenośnego, zasilanego z baterii.

Dotychczas istniejące układy zawierające elementy z efektem kwantowym, jak np. diody tunelowe, wykazywały pewne właściwości kwantowe, lecz ich wymiary i funkcje nie różniły się znacznie od wielkości klasycznych tranzystorów bipolarnych. Natomiast już pierwsze pokazy tranzystora, którego wszystkie zasadnicze elementy mają wymiary kwantowe, a działanie jest oparte na efektach kwantowych, wykazują, że mamy do czynienia z zupełnie nową generacją układów. Zwiększenie wydajności i zmniejszenie kosztów realizowanych funkcji w stosunku do stanu obecnego są tego samego rzędu, co zmiany wywołane przejściem z lamp próżniowych na półprzewodniki.

Tranzystor o efekcie kwantowym został opracowany przez wielu fizyków z ośrodka badawczego TI. Należałoby tu wymienić doktorów Marka Reeda, Williama Frensleya i Alana Seabaugha. Wyniki swych prac przedstawili w raporcie na Międzynarodowym Spotkaniu Urządzeń Elektronowych (IEMD – *International Electron Devices Meeting*) w San Francisco. Opisują oni element bipolarny, który może być traktowany jako prekursor tranzystorów kwantowych unipolarnych, jeszcze mniejszych i jeszcze szybszych, które mogą kiedyś dokonać przewrotu w całym przemyśle półprzewodników, a których obszary działania są rzędu 10–20 nm, a więc dziesięć tysięcy razy mniej niż średnica ludzkiego włosa.

W tym świecie quâsi-nieskończenie małym, wyjaśnia dr Reed, efekty kwantowe rządzą zachowaniem się materii i energii. Elektrony zajmują oddzielne i skoncentrowane poziomy energii, zwane pasmami, i wpadają w rezonans tak, jak głos w tunelu, o ile są ograniczone do wymiarów rzędu długości ich fali. W rezultacie otrzymuje się element umożliwiający przełączanie z wyjątkową dokładnością i wydajnością, z szybkością prawdopodobnie tysiące razy większą niż najlepsze półprzewodniki współczesne.

W klasycznych tranzystorach krzemowych mówi się o barierach potencjałowych materiałów półprzewodnikowych, które można poszerzać lub zwężać, zmieniając napięcie przyłożone do bazy, aby modulować przejście prądu między, emiterem a kolektorem. W tranzystorach kwantowych, różne poziomy energii charakteryzujące materiały użyte w bazie, emiterze i kolektorze, działają jako bariery na przepływ prądu. Prąd między emiterem a kolektorem popłynie tylko wówczas, gdy napięcie doprowadzone do bazy tranzystora będzie modulowane tak, by dokładnie odpowiadało tym poziomom energii i by elektrony mogły rezonować wywołując efekt „tunelowy" poprzecznie do bazy.

Dr. Reed podkreślił, że rewolucyjność tranzystora kwantowego polega na tym, iż udało się w końcu sterować bezpośrednio prąd poddany efektowi tunelowemu przez modulowanie napięcia studni kwantowej, czyli bazy. Poprzednio nie udawało się utrzymać bazy izolowanej od emitera i kolektora. Konstruowany obecnie tranzystor, który można porównać z tunelem rezonansowym, jest etapem w procesie opracowywania tranzystorów kwantowych unipolarnych, w których elektrony będą ograniczone do proporcji kwantowych w trzech wymiarach. Jednak opracowany przez Texas Instruments bipolarny tranzystor kwantowy, w którym zastosowano arsenek galu, aluminium i ind, przedstawia bardzo obiecujące właściwości.

Dyrektor zaawansowanych programów Centrum Badawczego – Robert Bate – podał, że wzmocnienie prądowe w normalnej temperaturze otoczenia jest rzędu pięćdziesięciu, a w niektórych egzemplarzach nawet dziewięciokrotnie większe. Szybkość przesyłania elektronów między emiterem a kolektorem jest tak duża, że trudno ją zmierzyć przy obecnym stanie techniki. Czas tego przejścia szacuje się na kilka femtosekund (10^{-12}s).

R. Bate powiedział, że choć samo opracowanie tranzystora kwantowego jest znaczącym krokiem w rozwoju dziedziny, którą można nazwać nanoelektroniką, bo zostało jeszcze wiele do zrobienia w dziedzinie wytwarzania i niezawodnego łączenia struktur tak małych przy produkcji w dużych ilościach; wymaga to bowiem technik, które jeszcze nie istnieją. Inną konsekwencją miniaturyzacji jest konieczność znalezienia nowych architektur układów.

Naukowcy z Texas Instruments pracują nad strukturami kwantowymi od 1982 roku. Pierwsze prace nad efektem tunelowym prowadzono też w laboratoriach badawczych marynarki i obrony USA. Pierwszy tranzystor kwantowy został opracowany w ramach kontraktu zawartego z Air Force Wright Aeronautical Laboratories (Aeronautyczne Laboratoria Sił Lotniczych USA im. Wrighta). Podobne prace prowadzi się w ramach projektu finansowego przez DARPA (Defense Research Projects Agency – Agencja Obronnych Projektów Badawczych).

Oprac. J. RYŻKO
na podstawie
minis & micros No. 314

Trzy systemy programowania w Moduli 2

W szerszym użyciu na komputerach zgodnych z IBM PC są obecnie trzy realizacje języka Modula 2: tzw. Native Code Modula-2, firmy Modula Corporation; kompilator Modula-2PC, firmy PCollier Systems i najbardziej znany system Modula-2/86, firmy Logitech. Wszystkie trzy realizacje tworzą w zasadzie systemy programowania, tzn. oprócz kompilatorów zawierają także narzędzia i programy usługowe do wspomagania pracy programistów.

Z niewielkimi wyjątkami, wszystkie te systemy są realizacjami pełnej składni języka Modula 2. Wszystkie zapewniają fundamentalną w Moduli 2 rozłączną kompilację modułów, z podziałem na moduły definicyjne, opisujące zewnętrzne cechy jednostki programowej i moduły implementacyjne, definiujące algorytmy i struktury danych, służące do wykonywania powierzonych zadań.

Wszystkie trzy pakiety wykonują podobnie trójstopniowe przekształcenie programu źródłowego na program wykonywalny. W pierwszym kroku program źródłowy jest kompilowany na plik wynikowy. Ponieważ program w Moduli 2 jest zwykle łączony z innymi modułami skompilowanymi rozłącznie, w drugim kroku używa się konsolidatora wyszukującego pliki wynikowe dla żądanych modułów i programu tworzącego pełny kod wynikowy. W trzecim kroku skonsolidowany program można wykonać używając polecenia *RUN*, podając jego nazwę (jeśli konsolidator tworzy plik o rozszerzeniu *EXE*) lub – za pomocą opcji konsolidatora.

We wszystkich pakietach istnieją skompilowane pierwotnie moduły biblioteczne, które mogą być importowane do programów deklaracją *FROM...IMPORT*. W większości są to moduły opisane w podręczniku standardowej wersji języka (N. Wirth, Modula 2, WNT, Warszawa, 1987), np. InOut, RealInOut, FileSystem, Storage, Terminal. Jednakże, oprócz tych modułów standardowych istnieje duża liczba innych modułów, różniących się znacznie co do nazw i przeznaczenia.

Ponadto wszystkie trzy systemy umożliwiają nakładkowanie, a także tworzenie i współpracę procesów współbieżnych oraz dostęp do systemu operacyjnego DOS za pomocą wywołań.

Native Code Modula-2

System firmy Modula Corporation ma dwie wersje kompilatora (jedną trzyczęściową dla komputerów o pamięci mniejszej niż 512 KB i drugą o wielkości 238 KB dla komputerów o większej pamięci) i pięć wersji konsolidatora. Tak duża liczba wersji konsolidatora wiąże się z tym, czy jest generowany kod dla koprocesora arytmetycznego i czy jest tworzony bezpośrednio plik typu *EXE*. Dwie wersje konsolidatora generują kod dla koprocesora 8087, a jedna z nich i jedna z dwóch wersji nie generujących kodu dla koprocesora tworzy pliki wykonywalne typu *EXE*. Piąta wersja konsolidatora łączy skompilowane rozłącznie moduły z modułami importowanymi w plik nie wykonywalny bezpośrednio, lecz umożliwiający szybsze załadowanie do wykonania. Choć liczba wymienionych wersji jest dość duża, w rzeczywistości skompilowanie programu i wygenerowanie pliku typu *EXE* wymaga zaledwie dwóch poleceń.

Podręcznik użytkowania systemu Native Code Modula-2 liczy 450 stron, zawiera sporo przykładów i jest bardzo czytelny, choć niezbyt łatwy. Opisano w nim instalowanie systemu zarówno na dyskach elastycznych, jak i na dysku stałym, przedstawiono zasady zgodności modułów, tzn. warunki ponownej kompilacji programu, oraz omówiono sygnalizację błędów. Najważniejszą cechą podręcznika jest szczegółowe scharakteryzowanie dostarczonych modułów bibliotecznych; wszystkie procedury w tych modułach są w pełni opisane i zilustrowane przykładami.

Modula-2PC

System Modula-2PC, złożony z kompilatora, konsolidatora i 16 modułów bibliotecznych,umożliwia utworzenie pliku typu *EXE* za pomocą dwóch poleceń. Nowsze wersje systemu zawierają też zintegrowany edytor, program uruchomieniowy i bibliotekę zmiennoprzecinkową. W przeciwieństwie do innych systemów, kompilator nie generuje komunikatów o błędach czasu wykonania, chyba że użyje się specjalnej opcji, a konsolidator standardowo nie wytwarza pliku *EXE*, lecz powoduje załadowanie modułów bibliotecznych i wykonanie programu wynikowego (istnieje oczywiście opcja konsolidatora umożliwiająca tworzenie pliku typu *EXE*).

Dokumentacja dostarczana wraz z systemem jest dość skąpa, choć wystarczająco dokładna. Zawiera instrukcję instalowania systemu na komputerach z jednym lub dwoma dyskami elastycznymi lub z dyskiem stałym. Większą część podręcznika zajmuje wprowadzenie do Moduli 2. Duża jego część jest poświęcona opisowi modułów bibliotecznych, choć bez wystarczającej liczby przykładów.

Modula-2/86

System Modula-2/86 firmy Logitech, w wersji 2, jest dostarczany w konfiguracji podstawowej, złożonej ze zintegrowanego edytora, dwóch wersji kompilatora i dwóch wersji konsolidatora (dla komputerów posiadających, odpowiednio, poniżej lub powyżej 512 KB pamięci operacyjnej), czterech plików instalacyjnych i 39 modułów bibliotecznych. Dla części systemu dostarcza się nawet kod źródłowy, na wypadek gdyby doświadczeni programiści pragnęli dostosować go do komputerów niezgodnych z IBM PC.

Praca w systemie może przebiegać standardowo, przy użyciu poleceń, lecz także – z wykorzystaniem zintegrowanego środowiska sterowanego przez menu, które jest bardzo wygodne w użyciu. Podstawą systemu zintegrowanego jest pełnoekranowy, wielookienkowy edytor, zoptymalizowany dla programów pisanych w Moduli 2. Oprócz typowych właściwości, takich jak automatyczne wcinanie wierszy, wycinanie i wklejanie tekstu, wywoływanie samouczka, edytor ma wbudowaną kontrolę składni Moduli 2, którą można uaktywnić w dowolnym momencie przez przyciśnięcie odpowiedniego klawisza funkcyjnego. W ten sam sposób można zresztą wywoływać kompilator i konsolidator, które po wykonaniu zadania przekazują sterowanie na powrót do edytora. Jeśli kompilator stwierdzi błąd w tłumaczonym programie, to edytor ustawia w tym miejscu kursor i wyświetla komunikat błędu w specjalnym oknie, o ile wciśnięto określoną kombinację klawiszy.

Inną ważną cechą edytora jest czułość na składnię. Każda z 26 kombinacji klawiszy od *ALT-A* do *ALT-Z* odpowiada jednej z konstrukcji składniowych, a po jej wywołaniu właściwym klawiszem, w miejscu programu oznaczonym kursorem następuje wstawienie odpowiedniego elementu składni. Przykładowo, wciśnięciu klawiszy *ALT-I* odpowiada wstawienie na miejsce kursora szkieletu instrukcji *IF*, tzn.:

```
IF ... THEN ...
END (*IF*);
```

i przesunięcie kursora za słowo zastrzeżone *IF*. Można również przedefiniować znaczenie poszczególnych klawiszy, w zależności od potrzeb.

Podręcznik użytkowania systemu jest, poza nielicznymi miejscami, bardzo przejrzysty i dobrze zorganizowany. Oprócz instrukcji instalowania i użytkowania systemu, zawiera opis modułów bibliotecznych w postaci wydruków modułów definicyjnych, opatrzonych licznymi komentarzami. Choć nie jest to opis tak wyczerpujący, jak w dokumentacji systemu Native Code Modula 2, to w większości zastosowań całkowicie wystarcza. Niektóre moduły biblioteczne są opisane dokładniej w innych częściach podręcznika.

Oprócz systemu podstawowego, firma Logitech dostarcza również szereg dodatkowych narzędzi, jak:

Make – automatycznie wykrywający współzależności między modułami i nieaktualne pliki wynikowe, i wytwarzający plik wsadowy z minimalną liczbą poleceń niezbędnych do ponownego uaktualnienia programu;

pakiet usługowy – zawierający statyczny program uruchomieniowy (ang. *post-mortem debugger*), disasembler, analizator odwołań zewnętrznych (ang. *cross-referencer*) i preprocesor o nazwie Version, który umożliwia kompilację warunkową;

pakiet Window – złożony z szeregu podprogramów bibliotecznych, umożliwiających korzystanie z wielu niezależnych okien;

dynamiczny program uruchomieniowy – pozwalający na ustawianie punktów wstrzymania, sprawdzanie i zmienianie danych podczas wykonywania programu;

biblioteka źródłowa – zawierająca kod źródłowy modułów bibliotecznych;

translator Turbo Pascala na Modulę 2.

Ostatnie z wymienionych narzędzi, translator Turbo Pascala, choć nie działa bezbłędnie dla wszystkich programów w Turbo Pascalu, to generuje dość poprawny kod, który można z łatwością poprawić ręcznie. Przykładowo, dostarczany w pakiecie Turbo Pascala przykładowy plik graficzny *ART.PAS* został przetłumaczony poprawnie z wyjątkiem dwóch wyrażeń w Pascalu zawierających zbiory znaków. Jedno z nich, mianowicie:

Upcase (Ch) in ['Y', 'N', # 27]

zostało przetłumaczone w Moduli 2 na następujące:

CAP (Ch) IN BITSET {'Y', 'N', 33C}

co jest prawie poprawne, lecz nie uwzględnia faktu, że typ *BITSET* w Moduli 2 nie pomieści zbioru znaków. Rozwiązanie tego problemu polega na zastąpieniu powyższego wyrażenia bezpośrednim obliczeniem wartości następującego wyrażenia logicznego:

(CAP (Ch) = 'Y' OR (CAP(Ch) = 'N') OR (Ch = 33C)

Teksty wzorcowe

Standardowe programy wzorcowe miesięcznika Byte, przetłumaczone z Basica na Modulę 2, uzupełnione o program 10000-krotnego wyprowadzenia na ekran litery „a" (w celu sprawdzenia szybkości obsługi ekranu), wykonane w trzech omówionych systemach, doprowadziły do otrzymania wyników przedstawionych w tabeli 1). Zapis na dysk i odczyt z dysku odnoszą się do 64-kilobajtowego sekwencyjnego pliku tekstowego i czystego dysku elastycznego. Obliczenia numeryczne dotyczą 10000 operacji mnożenia i tyluż operacji dzielenia liczb pojedynczej precyzji. Sito Erastotenesa odnosi się do dziesięciu iteracji. Wyniki testów określających precyzję typu rzeczywistego i znajdowanie największej i najmniejszej liczby rzeczywistej oraz – tzw. testu Dhrystone'a przedstawiono, odpowiednio, w tabelach 2 i 3. Choć ten ostatni program, opracowany przez R. P. Weickera i opublikowany w *Communications of the ACM* dla języka Ada, nie zawiera żadnych użytecznych obliczeń, to reprezentuje dość wiernie rzeczywistą mieszankę instrukcji spotykanych w praktycznych programach.

Biorąc pod uwagę szybkość i zwartość generowanego kodu, na podstawie przeprowadzonych testów najwyżej należy ocenić kompilator Modula-2/86. Jest on wolniejszy od pozostałych jedynie w przypadku obliczeń numerycznych, ale to tylko wskutek znacznie większej dokładności obliczeń zmiennoprzecinkowych w porównaniu z pozostałymi kompilatorami. Kompilator Modula-2/86 wytwarza najszybsze programy, co jest najbardziej widoczne w przypadku testu Dhrystone'a (tab. 3).

Choć kompilowanie i konsolidowanie programu w tym systemie trwa dłużej niż w innych, to tylko ten kompilator tłumaczył poprawnie test Dhrystone'a bez modyfikacji kodu. Dzieje się tak z trzech powodów.

Po pierwsze, kompilator Modula-2/86 jest kompilatorem wieloprzebiegowym, a pozostałe dwa są jednoprzebiegowe. O ile kompilatory jednoprzebiegowe są na ogół szybsze od wieloprzebiegowych, to muszą opracować deklarację procedury przed jej użyciem. W oryginalnym kodzie testu Dhrystone'a procedury były używane przed ich zadeklarowaniem, co powodowało błędy sygnalizowane przez oba kompilatory jednoprzebiegowe. Rozwiązanie tego problemu polega na zadeklarowaniu procedury z klauzulą *FORWARD*, podobnie jak w Pascalu.

Po drugie, żaden z obu gorszych kompilatorów nie rozpoznawał procedury *NEW*, którą należało zamienić na *ALLOCATE*.

Po trzecie, kompilator Modula-2PC nie przyjmował całkowicie poprawnej relacji porównania dwóch zmiennych typu *CapitalLetter*, zdefiniowanego uprzednio jako typ zawężony typu *CHAR*. Jest to oczywisty błąd kompilatora, który wyeliminowano przemianowując typ zawężony na *CHAR*.

Dokumentacja dla wszystkich trzech systemów pozostawia nieco do życzenia. Podręcznik systemu Native Code Modula-2 nigdzie nie wyjaśnia skutków kompilacji jednoprzebiegowej. Nie ma w nim nawet wzmianki o deklaracji *FORWARD*. W niektórych miejscach są natomiast zupełnie niepotrzebne odwołania do poprzednich wersji kompilatora. Podręcznik systemu Modula-2PC byłby lepszy, gdyby zawierał więcej przykładów użycia modułów

Tabela 1. Testy wzorcowe miesięcznika Byte dla programów w Moduli 2 (czas podano w sekundach)

Rodzaj testu	Native Code	Modula – 2PC	Modula – 2/86
Zapis na dysk elastyczny	161,30	159,89	160,00
Zapis na dysk stały	5,44	5,44	3,46
Odczyt dysku elastycznego	28,72	28,67	28,67
Odczyt dysku stałego	5,38	4,55	2,80
Obliczenia numeryczne	21,97	41,36	32,68
dokładność	1,19E-7	3,82E-8	1,11E-16
Sito Erastotenesa	19,00	31,03	16,65
Wyprowadzanie na ekran	37,57	37,52	31,31

Tabela 2. Test danych typu rzeczywistego

Parametr testu	Native Code	Modula – 2PC	Modula – 2/86
Rozmiar typu (B)	4	8	8
Największa liczba	3,4E+38	1,1E+37	1,7E+308
Najmniejsza liczba dodatnia	1,2E–38	2,9E–39	2,2E–308
Przybliżona precyzja (cyfry dziesiętne)	7 8	9 10	15 16

Tabela 3. Test Dhrystone'a (czas podano w sekundach, rozmiar plików – w bajtach; RTC ON/OFF – odblokowanie i zablokowanie kontroli czasu wykonania)

Parametr testu	Native Code	Modula – 2PC	Modula – 2/86
Czas kompilacji			
RTC ON	26,64	46,68	65,25
RTC OFF	23,84	45,52	65,91
Czas konsolidacji			
RTC ON	–	–	25,49
RTC OFF	–	–	26,20
Czas generacji pliku EXE			
RTC ON	8,78	7,80	22,91
RTC OFF	9,67	7,63	22,19
Całkowity czas tworzenia pliku EXE			
RTC ON	35,42	54,48	113,65
RTC OFF	33,51	53,15	114,30
Rozmiar pliku EXE			
RTC ON	73 728	55 296	44 160
RTC OFF	72 464	53 760	43 136
Szybkość (operacje Dhrystone'a/s)			
RTC ON	98,8	55,3	232,0
RTC OFF	88,7	53,2	150,0

bibliotecznych. Jego część zawierająca wprowadzenie do Moduli 2 jest dość dobrze napisana, ale zbyt skondensowana, aby można ją studiować samodzielnie bez uciekania się do pomocy raportu języka. W takim wypadku samo istnienie wprowadzenia traci więc sens. Z kolei, podręcznik systemu Modula-2/86 nie zawiera opisu niektórych używanych procedur standardowych.

Pod pewnymi względami wymienione kompilatory różnią się posiadaniem lub brakiem określonej cechy. Przykładowo, konsolidator Modula-2/86 jako jedyny nie posiada opcji bezpośredniego łączenia i wykonywania (ang. *load-and-go*); najpierw zachowuje moduł ładowalny na dysku i dopiero po oddzielnym poleceniu ładuje go ponownie i wykonuje. Modula-2/86 jest również jedynym systemem zawierającym moduł biblioteczny BCD, który umożliwia dokładne działanie (wraz z formatowaniem) na typowych monetarnych wielkościach liczbowych o maksymalnej długości 18 cyfr. Jest to również jedyny system umożliwiający pracę z procesorem 80286. Z kolei, system Modula-2PC, w tej wersji, nie umożliwia współpracy z programami napisanymi w języku asemblera (przynajmniej nie ma o tym żadnej wzmianki w dokumentacji). System Native Code Modula-2 jako jedyny posiada moduł biblioteczny grafiki kolorowej, zawierający podprogramy do rysowania kształtów. W systemie Modula-2/86 moduł taki jest dostępny tylko w pakiecie translatora Turbo Pacsala.

• • •

Podsumowując, należy stwierdzić, że najlepsze właściwości spośród trzech omawianych systemów ma Modula-2/86. Ten kompilator generuje najszybszy i najbardziej zwarty kod, akceptuje też pełny język Modula 2 zdefiniowany w raporcie N. Wirtha. Szczególnie użyteczny jest zintegrowany edytor, którego nie mają dwa pozostałe systemy. Modula-2/86 ma też największy zbiór modułów bibliotecznych, a jedynym istotnym niedostatkiem jest brak modułu graficznego. Ponadto, w skład tego systemu wchodzi opcjonalnie duża liczba programów usługowych.

W celu uzyskania pełnego obrazu jakości tych systemów, należałoby jeszcze poddać je porównaniu z innymi językami. Przykładowo, choć kompilator Modula-2/86 wytwarza względnie szybki i zwarty kod, to nie jest najlepszy w sensie bezwzględnym. Typowe kompilatory języka C generują bardziej zwarty kod, a przetłumaczone programy są wykonywane znacznie szybciej.

Oprac. **J. Zal.**

Przypominamy, że zlecenia na umieszczenia ogłoszeń w INFORMATYCE przyjmuje w bieżącym roku bezpośrednio nasza redakcja. Czekamy na zgłoszenia.

COIG

CENTRALNY OŚRODEK INFORMATYKI GÓRNICTWA
40–065 Katowice, ul. Mikołowska 100
tel. 574-777, teleks 00313711, telefaks 517-442

PROJEKTOWANIE I PROGRAMOWANIE SYSTEMÓW INFORMATYCZNYCH. KOMPLETACJA, SPRZEDAŻ SYSTEMÓW I SPRZĘTU KOMPUTEROWEGO, KSEROKOPIAREK, TELEFAKSÓW, PODZESPOŁÓW I CZĘŚCI ZAMIENNYCH.
INSTALACJA, NAPRAWY I PRZEGLĄDY URZĄDZEŃ MIKROKOMPUTEROWYCH.
SZKOLENIE PROGRAMISTÓW, PROJEKTANTÓW I OPERATORÓW SPRZĘTU INFORMATYCZNEGO.

Oferujemy usługi w zakresie ujmowania i przetwarzania danych w trybie wsadowo-partiowym oraz zdalnym realizowane na komputerach: MERA-9150, ODRA-1300, ICL-1900, 2900, 39, RIAD.
Sprzęt mikrokomputerowy kompatybilny z IBM PC konfigurujemy zgodnie z potrzebami użytkownika.
Instalujemy sieci lokalne. Tworzymy systemy aplikacyjne.
Prowadzimy serwis sprzętu mikrokomputerowego w zakresie instalacji, napraw i przeglądów.
Dostarczamy wymagane oprogramowanie narzędziowe.
Szkolimy z zakresu obsługi mikrokomputerów, systemów operacyjnych, sieci lokalnych i teletransmisji, języków programowania oraz generatorów baz danych.
Projektujemy systemy informatyczne wg założeń odbiorcy lub na podstawie analizy jego potrzeb.
Dostarczamy oprogramowanie gotowych projektów z zastosowaniem dowolnego języka.

EO/943/89

Oprogramowanie systemu sieciowego IBM

Masowe wykorzystanie komputerów osobistych, w szczególności komputerów typu PC XT i PC AT oraz ich licznych odpowiedników w systemach zarządzania, pracach biurowych, archiwach, magazynach, pracach badawczych wywołało naturalną konieczność łączenia ich w sieci.

Powstało wiele niezależnych koncepcji organizacji sieci. Jedną z nich jest włączenie różnego rodzaju sieci lokalnych, sterowanych komputerami PC XT, PC AT do standardu firmowego IBM nazywanego **Architekturą Systemów Sieciowych** *(Systems Network Architecture – SNA)* określającego zasady budowy sieci przy użyciu sprzętu i oprogramowania oferowanego przez IBM.

W tym celu opracowano program Net View/PC. Umożliwia on utworzenie jednolitego scentralizowanego systemu sieciowego ze standardowych elementów sprzęto-programowych IBM oraz elementów pochodzących od innych producentów. Zadaniem systemu jest przejmowanie i przetwarzanie wyselekcjonowanych informacji i sygnałów alarmowych z poszczególnych urządzeń sieci w celu usprawnienia wykonywania programów użytkowych.

W scentralizowanym systemie sieciowym IBM przewidziano następujące wyspecjalizowane struktury (sprzętowo-programowe) realizujące odrębne zadania:
– blok sterujący (nazwa firmowa – focal point) z pamięcią, analizujący i przetwarzający zebrane informacje, ustalający kolejność działań i śledzący ich wykonywanie, wyszukujący informacje, zbierający sygnały alarmowe systemu i uruchamiający niezbędne działania oraz wyświetlający informacje np. diagnostyczne o pracy systemu,
– blok wejściowy (nazwa firmowa – entry point), przejmujący informacje z urządzeń standardowych i przygotowujący je do przetworzenia przez blok sterujący,
– blok interfejsowy (nazwa firmowa – service point) umożliwiający dwustronną komunikację pomiędzy urządzeniami niestandardowymi a blokiem sterującym.

Program NET View/PC stanowi uzupełnienie oprogramowania mikrokomputerów osobistych typu PC XT i PC AT. Do jego realizacji niezbędny jest dysk twardy, pamięć RAM 640 K bajtów oraz karta interfejsowa tzw. Koprocesor Interfejsowy Czasu Rzeczywistego (Real-Time Interface Coprocesor – RTIC). Ponadto dla programu Net View/PC 1.0 wymagany jest również system operacyjny DOS 3.2 lub dalsze jego modyfikacje. Program Net View/PC zapewnia przede wszystkim realizację funkcji bloku sterującego (analiza, przetwarzanie, wyszukiwanie błędów, śledzenie pracy systemu, kolejkowanie zadań itp.) oraz bloku interfejsowego. Dodatkowo może on wykonywać firmowe programy obliczeniowe IBM.

Pełniąc funkcję bloku interfejsowego dużego systemu sieciowego program Net View/PC umożliwia zbieranie informacji z : firmowych zestawów urządzeń (CBXs), sieci lokalnych sterowanych komputerami PC XT/AT, multiplekserów (typ T_1) itp., przedstawionych w formacie PC i dostosowuje je do formatu SYSTEM/370. Informacje te są następnie przekazywane do bloku sterującego. Umożliwia to wzajemne połączenie standardowej aparatury firmowej i innej we wspólnej sieci.

Program Net View/PC może pełnić rolę bloku sterującego, przejmując niektóre funkcje programu komputera centralnego np. wyszukiwanie błędów, przyjmowanie, przetwarzanie i wyświetlanie sygnałów alarmowych itp. Strukturę programu przedstawiono na rysunku.

Podstawowymi elementami programu Net View/PC są: system główny i podprogramy komunikacji zewnętrznej. Zawiera on również podstawowe programy firmowe IBM (oznaczenie firmowe – ROLM), firmowe programy użytkowe, programy interfejsowe (nazwa firmowa application programming interface – APT). Program Net View/PC pracuje pod systemem operacyjnym CP/88 opracowanym dla IBM PC. Jest to system wielozadaniowy umożliwiający obsługę kilku użytkowników jednocześnie. Większość procedur DOS i BIOS wywołuje on kolejno lub emuluje. CP/88 umożliwia zwielokrotnienie seansów (tj. cykli pracy użytkownika z systemem – od wejścia do wyjścia z systemu). Każdy seans pozwala na korzystanie z klawiatury i monitora.

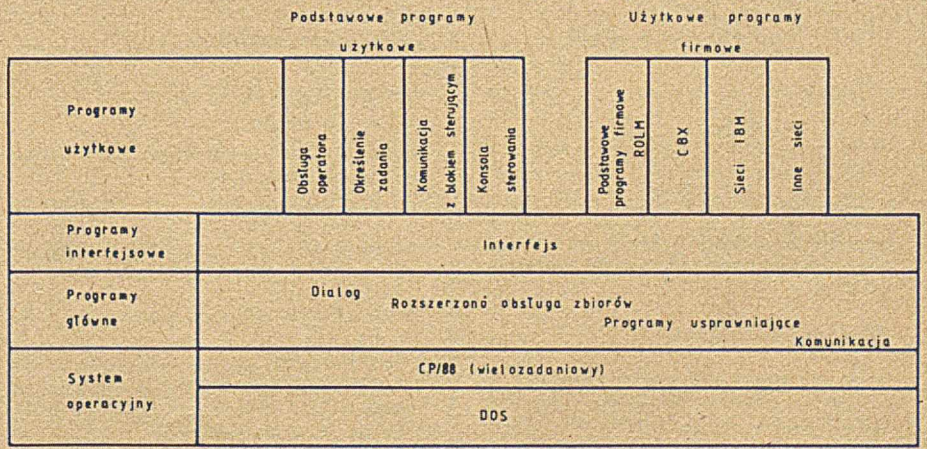
Ze względu na wielozadaniowość CP/88 jest w programie Net View/CP systemem nadrzędnym względem DOS. W celu wypełnienia wszystkich zadań przewidzianych w programie Net View/CP wiele różnych procesów musi być wykonywanych jednocześnie. Jeżeli np. odebrany zostanie sygnał alarmowy (wysłany przez jakikolwiek element systemu – sprzętowy lub programowy) informujący o wykrytym błędzie, musi być on sprawdzony, przesłany do bloku sterującego i umieszczony w bazie danych systemu. W czasie opracowywania tego sygnału, może pojawić się inny lub np.: nie zostanie zakończone opracowywanie poprzedniego. Ponadto w tym samym czasie Net View/PC może odbierać dane z bloku sterującego, wejściowego lub interfejsowego, a niezależnie od tych procesów użytkownik może sprawdzać potrzebne mu informacje.

System wykonuje również wiele dodatkowych zadań: ułatwia prowadzenie dialogu operator (zarządzający) – system, przeprowadza obsługę zbiorów itp.

Wszystkie podstawowe programy użytkowe Net View/PC wymagające ciągłego używania interfejsów mogą wykorzystywać podprogram dialogu, nazywany EZ-VU, wypełniający wiele funkcji firmowego programu dialogu systemu zarządzania (ang. *Interactive System Productivity Facility – ISPF*), łącznie z jego funkcjami pomocniczymi, jak np. testowanie klawiatury, monitora itp. Podprogram EZ-VU pozwala na znaczne odciążenie interfejsów i zwiększenie efektywności Net View/PC.

Program Net View/PC umożliwia rozszerzenie standardowej obsługi zbiorów przez DOS o funkcje łączenia, kolejkowania i wprowadzania. Rodzaje (typy) używanych zbiorów odpowiadają metodzie stosowanej do ich zapisu, pamiętania i dostępu. Różne zastosowania programu Net View/PC wymagają różnych typów baz danych. Ponieważ jednakże pokrywają się one częściowo ze sobą, obsługa bez danych jest przeprowadzana przez program główny i ogranicza się do tworzenia i kasowania zbiorów, ich archiwacji, modyfikacji, wyszukiwania i zapisu. Przewidziano również pewne funkcje umożliwiające przeglądanie.

Oprócz przedstawionych ułatwień w prowadzeniu dialogu i obsługi zbiorów system ma możliwość rozbudowy obsługi i uprawnienia wykonywania programów użytkowych oraz możliwość sygnalizacji stanu (ang. *status line*) oraz generacji piktogramów (ang. *icons*).



Struktura programu Net View/PC

Usprawnienie wykonywania programów wymagających np. komunikacji międzyprocesowej polega na wyszukiwaniu identyfikatorów procesu (na podstawie nazwy tablicy) związanego z innym programem.

Inną ważną zaletą programu Net View/PC jest sygnalizacja stanu. Linia stanu (25) wyświetlana na monitorze jest uaktualniana i odświeżana w każdym seansie. Wskazuje ona np. jakie działania powinien podjąć operator, aby poprawić błędnie wprowadzone zbiory oraz umożliwia korzystanie z piktogramów.

Pełniąc rolę bloku interfejsowego program Net View/PC musi zapewnić komunikację pomiędzy standardowymi urządzeniami systemu i urządzeniami pozastandardowymi. Do bezpośredniej łączności urządzeń standardowych SNA z blokiem sterującym wykorzystywany jest synchroniczny protokoł transmisji SDLC (ang. *Synchronous Data Link Control*). Do komunikacji z urządzeniami spoza standardu SNA, innych standardów IBM (np. CBX_s) oraz innych urządzeń przewidziano transmisję asynchroniczną z wykorzystaniem karty Koprocesora Interfejsowego Czasu Rzeczywistego – RTIC.

Program Net View/PC stanowi ważne uzupełnienie oprogramowania komputerów osobistych PC XT i AT i może być szczególnie użyteczny w ośrodkach nasyconych aparaturą IBM, umożliwiając tworzenie bardzo elastycznych zestawów przez pełne wykorzystanie właściwości PC i standardowej sieci IBM.

KRZYSZTOF RZYMKOWSKI

Bardzo prosimy Autorów przysyłających teksty artykułów do opublikowania w INFORMATYCE o podawanie danych umożliwiających bezpośredni kontakt w godzinach pracy (numer telefonu) oraz adresu domowego i ewentualnie numeru konta bankowego.

Kardasiewicz J.: System MikroAPO. Podsystem projektowania technologii dla typoszeregów części

INFORMATYKA 1990, nr 4, s. 1

Charakterystyka podsystemu projektowania technologii części maszyn na mikrokomputerach klasy IBM PC, wchodzącego w skład kompleksowego systemu MikroAPO, oryginalnego polskiego rozwiązania w dziedzinie wspomaganego projektowania maszyn.

Kardasiewicz J.: MikroAPO system. A subsystem for technology design of spare part series

INFORMATYKA 1990, No. 4, p. 1

Characteristics of the subsystem for machine part technology design on microcomputers of the IBM PC category, which is a component of the MikroAPO integrated system, an original polish solution for computer assisted machine design.

Kardasiewicz J.: MikroAPO-System. Ein Untersystem für Projektierung der Maschinenteiletechnologie

INFORMATYKA 1990, Nr 4, S. 1

Eine Charakteristik des Untersystems für Projektierung der Maschinenteiletechnologie mit Anwendung der Computer von IBM PC-Klasse, das zum integrierten MikroAPO-System gehört und eine originale polnische Lösung aus dem Bereiche der rechnerunterstützten Maschienentechnologieprojektierung repräsentiert.

Bluemke I.: Architektury procesorów tablicowych tolerujących błędy

INFORMATYKA 1990, nr 4, s. 5

Przegląd najbardziej interesujących rozwiązań konstrukcyjnych procesorów tablicowych tolerujących błędy oraz określenie kryteriów porównywania takich procesorów.

Bluemke I.: Architectures of fault tolerant array processors

INFORMATYKA 1990, No. 4, p. 5

An overview of the most interesting solutions of fault tolerant array processors and definition of comparison criteria for such processors.

Bluemke I.: Architekturen der fehlertoleranten Parallelprozessoren

INFORMATYKA 1990, Nr 4, S. 5

Ein Überblick der am interessantesten Konstruktionslösungen von fehlertoleranten Parallelprozessoren und eine Definierung der Kriterien für Vergleichung solcher Prozessoren.

Czepielewski S., Nowakowsl Z.: Środowisko programowe do projektowania pakietów elektronicznych systemów mikroprocesorowych

INFORMATYKA 1990, nr 4, s. 11

Charakterystyka najczęściej stosowanych programów wspomagających projektowanie pakietów sprzętowych dla mikrokomputerów klasy IBM PC oraz omówienie sposobu wykorzystania i wyników testowania tego oprogramowania.

Czepielewski S., Nowakowski Z.: Software environment of hardware packages design for microprocessor systems

INFORMATYKA 1990, No. 4, p. 11

Characteristics of the most applied programs for IBM PC hardware packages assisted design and discussion of such software application methods and testing results.

Czepielewski S., Nowakowski Z.: Programmumwelt der Hardwarepaketeprojektierung für Mikroprozessorsysteme

INFORMATYKA 1990, Nr 4, S. 11

Eine Charakteristik der am meisten angewendeten Programme der rechnerunterstützten Hardwarepaketeprojektierung für Mikrocomputer der IBM PC-Klasse und eine Besprechung der mit solcher Software verbundenen Anwendungsmethode und Testungsergebnisse.

Zalewski J.: Okres istnienia oprogramowania (3)

INFORMATYKA 1990, nr 4, s. 14

Trzecia część omówienia norm amerykańskich dotyczących organizacji procesu wytwarzania produktów programowych.

Zalewski J.: Software life cycle (3)

INFORMATYKA 1990, No. 4, p. 14

Third part of american standards for organization of program products manufacturing process.

Zalewski J.: Lebenszyklus der Software (3)

INFORMATYKA 1990, Nr. 4, S. 14

Dritter Teil einer Besprechung von amerikanischen Normen aus dem Bereich der Organisation des Programmprodukterzeugungsprozesses.

Banachowski L., Deminet J., Lao M. J.: Propozycja nowego programu uniwersyteckich studiów informatycznych (3)

INFORMATYKA 1990, nr 4, s. 21

Trzecia część szczegółowego omówienia propozycji nowego programu uniwersyteckich studiów informatycznych w Polsce, wzorowanego na rozwiązaniach zalecanych przez wiodące światowe organizacje informatyczne.

Banachowski L., Deminet J., Lao M. J.: A proposal for a new program of informatics university studies (3)

INFORMATYKA 1990, No. 4, p. 21

Third part of detailed discussion of the proposal for new informatics university studies program in Poland, which is modelled after solutions recommended by leading informatics worldwide organizations.

Banachowski L., Deminet J., Lao M. J.: Ein Vorschlag des neuen Programmes für Informatik-Universitätsstudien (3)

INFORMATYKA 1990, Nr 4, S. 21

Dritter Tei einer detailierten Besprechung des Vorschlages von neuen Programm für Informatik Universitätsstundien, das nach Empfehlungen der führenden Informatikweltorganisationen erarbeitet wurde.

Atrybuty jakości oprogramowania – definicje podręcznikowe

W tym odcinku rubryki terminologicznej podaję ostatni już zestaw definicji atrybutów jakości oprogramowania, oparty na podręczniku: J. Vincent, A. Waters, J. Siclair; *Software Quality Assurance – Practice and Implementation*. Prentice-Hall, 1988.

Dokładność (ang. *accuracy*) – osiągnięcie wymaganej precyzji obliczeń pośrednich i wyników.

Dostępność (ang. *acces control and audit*) – możliwość sterowania dostępem i lustrowania dostępu do oprogramowania i danych.

Efektywność (ang. *efficiency*) – ilość zasobów komputerowych i objętość kodu wymagane przez program do wykonania swoich funkcji.

Elastyczność (ang. *flexibility*) – miara wysiłku wymaganego do zmodyfikowania eksploatowanego programu.

Integralność (ang. *integrity*) – stopień, w jakim można kontrolować dostęp do oprogramowania lub danych przez nieupoważnione osoby.

Komunikatywność (ang. *communicativeness*) – stopień przyswajalności danych wejściowych i wyników działania oprogramowania.

Konserwowalność (ang. *maintainability*) – miara wysiłku wymaganego do umiejscowienia i usunięcia defektu w eksploatowanym programie.

Łatwość obsługi (ang. *training*) – podatność oprogramowania na opanowanie przez użytkownika.

Modularność (ang. *modularity*) – posiadanie budowy mającej charakter zestawu maksymalnie niezależnych modułów.

Niesprzeczność (ang. *consistency*) – jednolitość technik i notacji projektu i implementacji.

Niezależność (ang. ⟨*software system, machine*⟩ *independence*) – zdolność oprogramowania do wykonywania swych funkcji bez względu na środowisko programowe lub sprzęt.

Niezawodność (ang. *reliability*) – stopień wiarygodności z jaką program wykonuje swoje funkcje z wymaganą precyzją.

Ogólność (ang. *generality*) – miara zakresu wykonywanych funkcji.

Poprawność (ang. *correctness*) – stopień w jakim program spełnia swoją specyfikację i odpowiada potrzebom użytkownika.

Prostota (ang. *simplicity*) – zaimplementowanie funkcji w najbardziej zrozumiały sposób.

Przenośność (ang. *portability*) – miara wysiłku wymaganego do przeniesienia programu z jednej konfiguracji sprzętowej lub środowiska do innej konfiguracji lub środowiska.

Rozszerzalność (ang. *expandability*) – możliwość zwiększenia pojemności pamięci lub ilości funkcji obliczeniowych.

Samoopisywalność (ang. *self-descriptiveness*) – wyjaśnianie sposobu zaimplementowania funkcji.

Sondowalność (ang. *instrumentation*) – możliwość wykonywania pomiarów użycia określonych konstrukcji lub – identyfikowania błędów.

Standardowość (ang. ⟨*communication, data*⟩ *commonality*) – stopień wykorzystania standardowych protokołów, sprzężeń i reprezentacji danych.

Szybkość (ang. *execution efficiency*) – czas wykonywania kodu.

Śladowalność (ang. *traceability*) – podatność oprogramowania na odtworzenie jego rozwoju od formułowania wymagań do implementacji, w określonym środowisku produkcyjnym i eksploatacyjnym.

Testowalność (ang. *testability*) – miara wysiłku wymaganego do przetestowania programu w celu upewnienia się, że wykonuje on pożądane funkcje.

Tolerowanie błędów (ang. *error tolerance*) – ciągłość działania w warunkach odbiegających od nominalnych.

Użyteczność (ang. *usability*) – miara wysiłku wymaganego do nauczenia się obsługi, eksploatowania, przygotowywania danych wejściowych i interpretowania wyników działania programu.

Używalność (ang. *operability*) – stopień określoności procedur eksploatowania oprogramowania.

Wielokrotność (ang. *reusability*) – stopień, w jakim program może być używany w innych zastosowaniach związanych z zakresem jego funkcji i obudowaniem.

dokończenie na s. 23

PRZEDSIĘBIORSTWO
ZASTOSOWAŃ INFORMATYKI

meditronik

oferuje:
- systemy mikrokomputerowe
- programy aplikacyjne dla różnych dziedzin gospodarki (na życzenie wysyłamy katalog)
- poszukiwane komponenty elektroniczne
- interfejs do kamery video (opc. CCD) z bogatą biblioteką oprogramowania
- emulator Z80
- tester układów scalonych i pamięci
- programator BPROM
- asynchroniczny procesor komunikacyjny
- konwerter RS-232 – Centronics

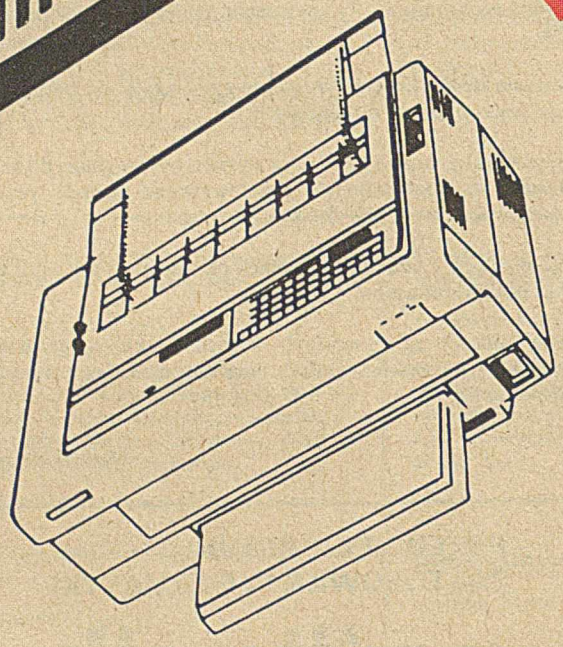
instaluje:
- połączenia międzykomputerowe (XT/AT – ODRA/RIAD/IBM)
- systemy sieciowe (NOVELL)
- systemy wielodostępne (SCO Xenix 286, 386, Unix System V)

Jeżeli jesteś autorem oryginalnego programu aplikacyjnego
- skontaktuj się z nami, będziemy pośredniczyć w sprzedaży Twojego programu, dbając o ochronę Twoich praw autorskich!

Nasz adres:
00–194 Warszawa, ul. Dzika 4
tel. (02) 635-22-63, 635-22-64
fax (02) 635-21-95
tlx 816075 medi pl

EO/605/89

SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA
SEIKOSHA



Japoński koncern SEIKO produkujący znane zegarki SEIKO, migawki do fotoaparatów stał się największym dostawcą drukarek komputerowych na rynku europejskim.

Informacje techniczne, katalogi:
SOFT-TRONIK SERVICE
00–710 Warszawa,
ul. Idzikowskiego 6
tel. (022) 40-46-79, (02) 642-46-79,
(02) 642-26-26, (02) 642-36-36,
telefaks: (02) 642-22-36,
teleks: 816075

EO 717 89

EO/717/89